

DOS EXTRA

18,- DM 145 Ös 18,- sfr

Sonderausgabe DOS EXTRA

Nr. 6 1989

ISSN 0934 - 2842
DMV-Verlag

Eine Publikation der
DOS INTERNATIONAL

Basic im Detail

- Event-Handling
- Dateiverwaltung
- Rekursion

GW-Basic intern

- Speicherverwaltung
- Variablentypen
- Token-Format

Turbo-Tempo mit SBasic Ein Compiler zum Abtippen

Utilities der Extraklasse

- GW-Basic verwaltet
dBase-Daten
- Basic selbst erweitert

Seitenweise Tips & Tricks
für GW-Basic



Hier ist die ideale Ergänzung

InfoWare

für Anwendungsprogramme

Einfacher geht's nicht!

Bei jeder Anwendung tauchen Fragen auf.
InfoWare hat stets die Antwort.
Ohne wenn und aber, ohne Ihr Programm zu verlassen.
Bei Bedarf können zusätzliche Erklärungen abgerufen werden.
Hunderte von Begriffen pro Titel werden verständlich und umfassend erklärt.

Was Ihnen nur InfoWare bieten kann:

Selbst erarbeitete Lösungswege nimmt InfoWare auf und integriert sie in sein Informationssystem.
So schaffen Sie sich Ihr persönliches Lexikon, das immer auf dem neuesten Stand ist.



Technische Details:

- Speicherresident
- Aufruftaste änderbar
- Destallprogramm zum Entfernen von Infoware aus dem Hauptspeicher
- Pulldowntechnik
- gezieltes Suchen nach Begriffen
- Datenexport in Ihre Anwendung
- Datenimport aus Ihrer Anwendung für eigene Begriffe
- um 50 eigene Begriffe erweiterbar

Notwendige Konfiguration:

IBM und kompatible PC XT/AT mit 512 kByte oder mehr, 1 Diskettenlaufwerk bzw. Festplatte, PC-DOS/ MS-DOS ab Version 2.0, mit Monochrom- oder Farbbildschirm

Die mit * gekennzeichneten Produkte sind eingetragene Warenzeichen der Firmen Microsoft, Borland, Ashton Tate, Wordperfect-Corporation, Fox Software Inc., Nantucket, Star-Division und SPI



Lieferbare Titel:

MS-DOS°

Ihr Problem mit MS-DOS: Sie möchten Ihre Daten sichern und wissen den entsprechenden Befehl nicht. InfoWare per Tastendruck aufrufen. Sofort wird der Befehl "BACKUP" umfangreich mit der entsprechenden Syntax erklärt. Übergabefunktion auslösen; Befehl wird ausgeführt.

PC-DOS°

GW-Basic°

Turbo-Basic°

Quick Basic°

Quick C°

Turbo C°

PC-Grundlagen°

Assembler 8086/88°

C-Grundlagen°

MS-Word 3.0/4.0° Grundlagen

WordPerfect 4.2° Grundlagen

Lotus 1-2-3°

Folgende Titel erscheinen in Kürze

dBase III plus°
Clipper°
Fox Base°
Multiplan°
Turbo Pascal 4.0°

Modula II
Star-Writer°
MS-Word°-Fortgeschrittene
WordPerfect°-Fortgeschrittene
Open Access II°

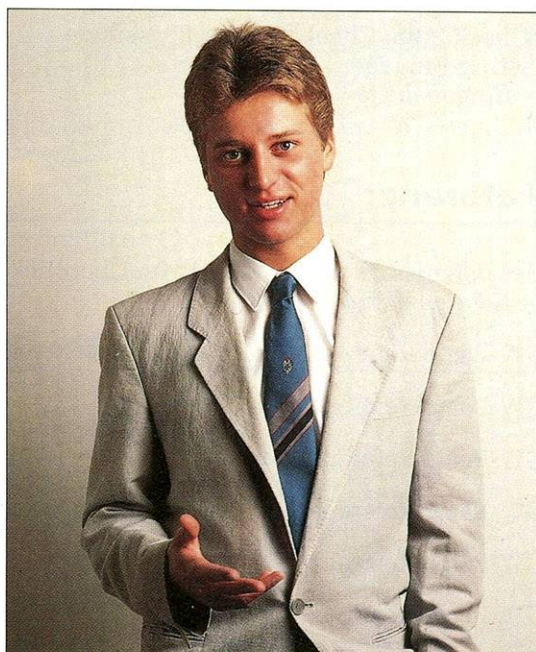
Noch immer nicht überzeugt?
Dann sollten Sie unser InfoWare Demo-Diskette für nur 5,- DM anfordern (Vorkasse)

InfoWare ist für nur **69,- DM*** pro Titel erhältlich

DMV-Verlag · Abt. Versand
Postfach 250 · 3440 Eschwege
Händleranfragen erwünscht

* Unverbindliche Preisempfehlung - Bei Bestellung über unsere Mailorder gilt folgendes: Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland 3,- DM bzw für das Ausland 5,- DM Porto und Verpackung
(c) 1988 DMV, Eschwege Licensed by infoPRO, infoPRO ist ein Warenzeichen der HOMESoft GmbH & Co.KG, München

Basic für Einsteiger und Profis



Programmieren Sie in Basic oder haben Sie vor, Programme in Basic zu schreiben? Wer sich bereits ernsthaft mit Basic beschäftigt hat, weiß diese Sprache zu schätzen, denn in ihr lassen sich kleine Programmideen sofort und ohne viel Zeitaufwand verwirklichen. Doch Basic kann noch viel mehr, und das möchten wir Ihnen mit diesem DOS Extra beweisen.

Gleichgültig, ob Sie erst mit Basic anfangen wollen, gelegentlich in Basic programmieren, fortgeschrittener Basic-Programmierer sind oder sich schon zu den Basic-Profis zählen: Diese Ausgabe des DOS Extra bietet Ihnen in jedem Fall eine Menge an neuen Informationen und Programmierhilfen.

Wenn Sie gerade in die Programmiersprache Basic frisch einsteigen, vermitteln Ihnen ausführliche Beiträge Grundlagen zu verschiedenen Themen wie Grafik, Sound oder Dateiverwaltung. Diese Beiträge sind mit Beispielen garniert und helfen Ihnen bei der Realisierung eigener Ideen. Gehören Sie zu den fortgeschrittenen Programmierern, dann werden die Referenztabellen für Sie sehr hilfreich sein. Hier finden Sie ohne Zeitverzug genau den Befehl, den Sie suchen – inklusive der erforderlichen Syntax. Wer sich mit Basic bereits sehr gut auskennt, aber tiefer einsteigen möchte, erfährt in der Rubrik »Basic intern« Wissenswertes über das GW-Basic-interne Token-Format, über den Aufbau von Programmzeilen, das Speicherformat von Variablen und so weiter.

Für den Fall, daß Sie konkrete Anwendungen bevorzugen, bieten wir Ihnen eine interessante Auswahl einsatzbereiter Applikationen – von der kleinen, rekursiven Mathematik-Spielerei bis hin zur komfortablen, menügesteuerten Datenbank.

Natürlich dürfen auch die Tips und Tricks wieder nicht fehlen. Unter anderem erfahren Sie, mit welchen Kniffen Sie die Ausführungsgeschwindigkeit Ihrer Basic-Programme signifikant erhöhen können.

Eine kleine Sensation ist der Basic-Compiler »SBasic« zum Abtippen, der viele GW-Basic-Programme problemlos in schnelle, direkt ausführbare Exe-Dateien übersetzt. Und was brauchen Sie, um SBasic benutzen zu können? Nur das, was sowieso jeder PC-Besitzer hat: GW-Basic! Mit GW-Basic können Sie nämlich den Quelltext von SBasic zum einsatzbereiten Compiler übersetzen. Und dann sehen Sie auch gleich, wie gut SBasic ist. Wenn Sie nämlich die Zeit, die GW-Basic zum Erzeugen des SBasic-Compilers benötigt, mit der Zeit vergleichen, die SBasic nach der Compilierung braucht, um sich selbst zu übersetzen, dann werden Sie eine Geschwindigkeitssteigerung um den Faktor 17 und mehr feststellen.

Mehrere Programme dieser Ausgabe lassen sich mit SBasic compilieren und auf diese Weise erheblich beschleunigen, so unter anderem auch die Datenbank.

Daß GW-Basic und SBasic sich sogar erweitern lassen, beweisen zwei Maschinencode-Module, die zusätzliche String-Funktionen und fensterorientierte Befehle zur Verfügung stellen.

Alles in allem ist dieses DOS Extra für alle PC-Besitzer, die in Basic programmieren oder vorhaben, mit dieser Sprache zu arbeiten, eine Fundgrube an Informationen und – wie wir meinen – unverzichtbar.

Ihr

Martin Aschoff

Eingabehilfe:

SCheck 2.0b: Checksummer für sichere Listing-Eingabe	6
- Eingabehilfe	

Referenz:

Der Befehlssatz von GW-Basic im Überblick	10
- Referenztabelle	
GW-Basic-Referenz: Programmablauf-Steuerung	16
GW-Basic-Referenz: Grafik	18
GW-Basic-Referenz: Sound	21
GW-Basic-Referenz: Ein- und Ausgabe	22
GW-Basic-Referenz: Variablen	26
GW-Basic-Referenz: Arithmetik	27
GW-Basic-Referenz: Event-Handling	29

Grundlagen:

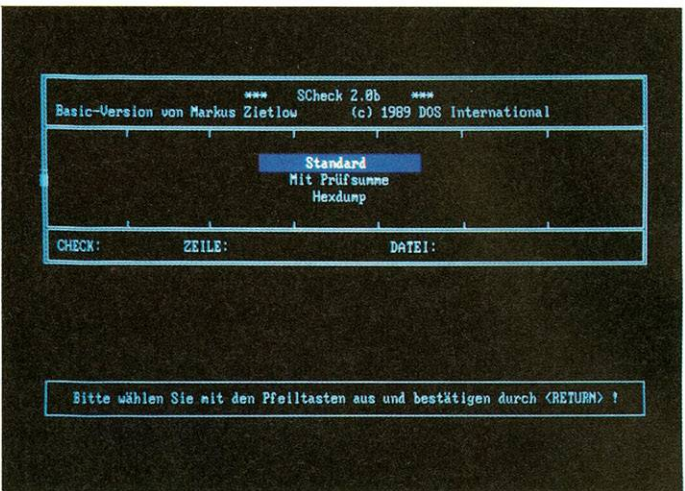
Grafikprogrammierung mit GW-Basic	32
- Der Draw-Befehl und seine Verwandten	
Sound-Programmierung auf dem PC	44
- Tonerzeugung im Detail	
Dateiverwaltung mit GW-Basic	50
- Die verschiedenen Dateitypen	
Rekursion und GW-Basic	55
- Rekursive und iterative Programmstrukturen	
Unterbrechen Sie mich bitte!	60
- Ereignisbehandlung in GW-Basic	
Basic von A bis Z	68
- Lexikon	

Quick Basic:

Strukturierte Programmierung mit Quick Basic 4.0	72
- als Beispiel: Superhirn	

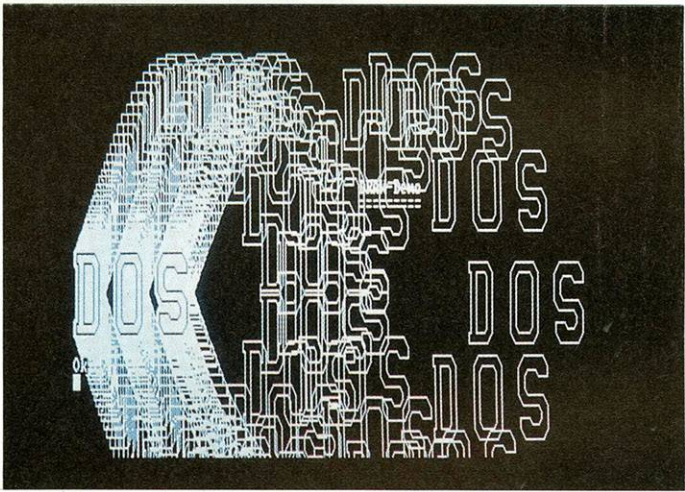
Utilities:

Die Kartenfrage	92
- Abfrage der Grafikkarte	
dBase-Datenbanken in Basic weiterverarbeiten	95
- Umsetzungsprogramm	
16 Farben auf CGA mit Turbo Basic	98
- Funktionsbibliothek	



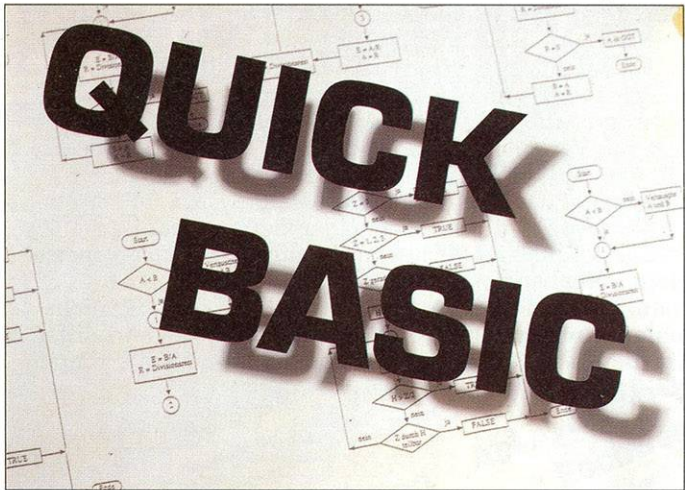
Mit der neuen Basic-Version 2.0b von SCheck sind alle Risiken bei der Listingeingabe ausgeräumt. Das Listing finden Sie auf

Seite 6



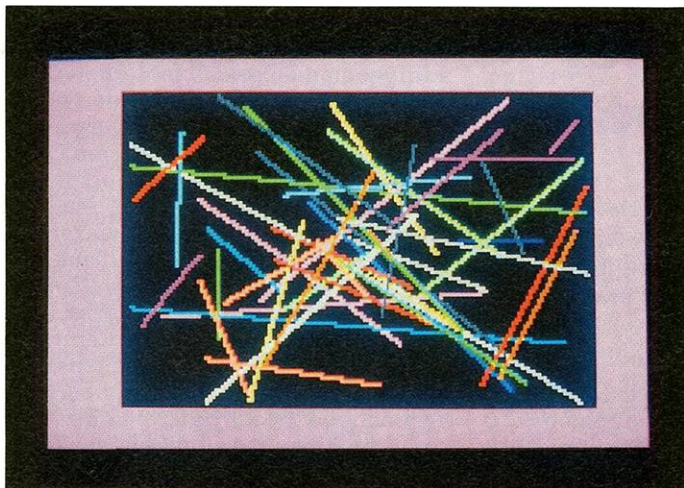
Wird der Grafik-Befehl »draw« von GW-Basic auch oft unterschätzt, ist er doch zu beachtlichen Leistungen fähig

Seite 32



Lernen Sie die Raffinessen der strukturierten Programmierung in Quick-Basic 4.0 kennen, mehr darüber ab

Seite 72



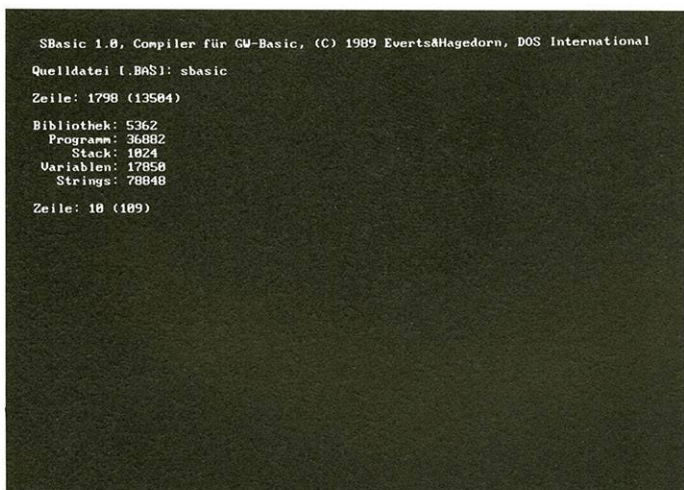
Das Turbo-Basic-Listing »cga.inc« zaubert auf einer CGA-Farbkartengrafikkarte im Grafikmodus 16 Farben auf Ihren Bildschirm

Seite 98



Mit dem Listing »Parat« erhalten Sie ein komplettes Datenbanksystem mit viel Komfort – und trotzdem in Basic

Seite 110



Diese Ausgabe des DOS Extra präsentiert Ihnen SBASIC, einen Basic-Compiler zum Abtippen

Seite 138

Basic-Erweiterungen:

Fensterorientierte Funktionen selbst programmiert	104
– mit Programmlisting	
Dauerhafte Funktionstastenbelegung in GW-Basic	106
– GW-Basic gepatcht	
Basic-Funktionen zur String-Verwaltung	108
– mit Programmlisting	

Anwendungen:

Mit »Parat« alle Daten am Start	110
– Datenbank	
Maskengenerator	118
– Alternative zum Print-Befehl	

Basic Intern:

Die Speicherverwaltung von GW-Basic	120
Variablen speichern unter GW-Basic	122
Basic-Programm selbstgemacht	127
– oder Über das interne Format von Basic-Befehlen	

Tips & Tricks:

Beschleunigung von Basic-Programmen	130
Tips und Tricks	132

Compiler:

SBASIC – Ein Compiler zum Nulltarif	138
– Listing	

Rubriken:

Basic für Einsteiger und Profis	3
– Editorial	
Sie haben uns etwas zu sagen	89
– Wettbewerb	
Zum Thema Listings	91
– ohne Spielregeln geht es nicht	
Impressum	170
Inserenten	170

SCheck 2.0b: Checksummer für sichere Listing-Eingabe

Die Programme dieses DOS Extra werden – wie auch alle Listings in der DOS International – nach einem speziellen Verfahren mit Prüfsummen versehen und abgedruckt. Das für die Überprüfung notwendige Prüfsummenprogramm »SCheck« liegt jetzt auch als Version 2.0b in Basic vor.

Alle in der DOS Extra und der DOS International abgedruckten Programme sind mit Prüfsummen versehen. Jede Zeile beginnt mit dieser dreistelligen Zeichenfolge. Sie ist in eckige Klammern gesetzt und wird beim Abtippen nicht mit eingegeben. Nach der Klammer folgt ein Leerzeichen, und darauf der eigentliche Text der Programmzeile. Nur diesen letzten Teil der abgedruckten Zeile geben Sie beim Abtippen ein.

Bei allen Programmzeilen wird im Abdruck nach jeweils 55 Zeichen ein Zeilenumbruch vorgenommen. Die umgebrochene Teilzeile erhält dabei keine Prüfsumme. Bei der Eingabe geben Sie umgebrochene Zeilen in einem Stück ein, drücken also nicht die Return-Taste wenn ein Zeilenumbruch im Ausdruck steht, sondern nur, wenn eine neue Zeilennummer auftaucht.

Assembler-Listings und hexadezimale Eingaben werden mit spezieller Zeilennummer wiedergegeben. In diesem Fall dürfen Sie die Programmzeile nur ohne Prüfsumme und ohne Zeilennummer eingeben. Diese Zeilennummer wird mit einem Doppelpunkt abgeschlossen. Der eigentliche Text beginnt nach dem folgenden Leerzeichen.

In der DOS International gilt das zuletzt Gesagte für alle Nicht-Basic-Programme. Verlangt der Interpreter oder Compiler keine Zeilennummer, so wird der Übersicht wegen beim Druck eine Numerierung hinzugefügt. Die Zeilennummer darf aber in keinem Fall miteingegeben werden.

Beim Abtippen von Programmen sind Tippfehler kaum zu vermeiden. Mit »SCheck« überprüfen Sie die abgetippten Listings auf eventuelle Fehler. Das Programm erlaubt nicht nur die Diagnose von fehlerhaft eingegebenen Zeilen, sondern mittels eines integrierten Editors auch gleich die Korrektur der fehlerhaften Zeilen. SCheck wird unter GW-Basic mit

run "scheck"

gestartet. Als erstes wählen Sie die gewünschte Prüfsummenversion. In der vorliegenden DOS Extra brauchen Sie immer »SCheck Version 2.0«. Nur wer Programme aus Ausga-

ben der DOS International vor Oktober 1988 eingeben will, muß den ebenfalls integrierten Prüfsummenalgorithmus »SCheck Version 1.0« aufrufen. Danach geben Sie den gewünschten Prüfmodus – normalerweise »Standard« – an und abschließend den Programmnamen des Listings, so wie Sie es auf dem Datenträger abgelegt haben.

Zuvor sollten Sie sich aber sicherheitshalber eine Kopie Ihres Listings anfertigen. SCheck arbeitet zwar nicht mit dem Original, sondern mit einer temporären Zwischendatei und verändert die Originaldatei erst ganz zum Schluß, nach dem erfolgreichen Abschluß der Arbeit. Dennoch sollten Sie auch ein noch so geringes Risiko eines Datenverlusts, etwa durch Fehler beim Speichern, ausschalten. Wie Sie im Bild sehen, zeigt SCheck die aktuelle Zeile stets in einem besonderen Bildschirmfenster an. Die Prüfsumme erscheint links unter dem Fenster. Beim Überprüfen Ihres Listings vergleichen Sie die angezeigte Prüfsumme mit der in der DOS Extra in eckigen Klammern abgedruckten Prüfsumme. Rechts daneben finden Sie die aktuelle Zeilennummer und den Dateinamen. Im unteren Fenster werden Fehlermeldungen und Hinweise auf die weitere Bearbeitung ausgegeben. Im abgebildeten Fall können Sie mit [Return] die Zeile übernehmen oder mit [E] den Editor aufrufen.

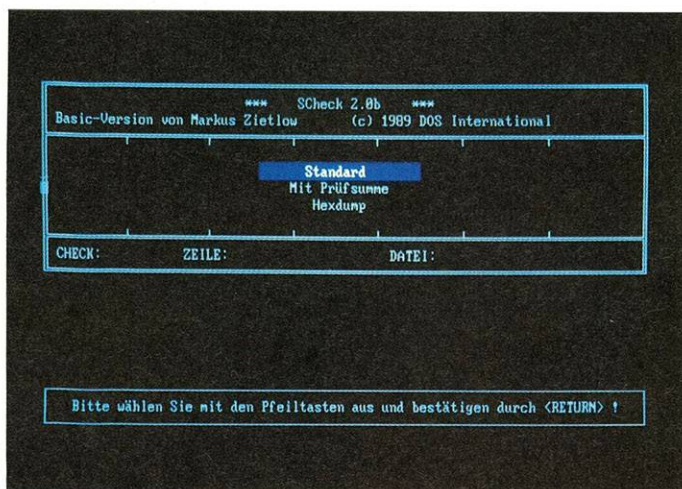
Der Editor dient der Korrektur fehlerhafter Zeilen. Er arbeitet immer im Einfügemodus, so daß fehlerhafte Eingaben extra gelöscht werden müssen. Sie können also nichts versehentlich überschreiben. Mit den Cursor-Tasten steuern Sie den Cursor im Editierfenster. [Home] bringt den Cursor an den Anfang der angezeigten Zeile, [End] entsprechend ans Ende. Löschen können Sie das Zeichen links vom Cursor mit der großen Löschtaste oberhalb der Return-Taste, [Del] entfernt das Zeichen unter dem Cursor. Mit [Return] übernehmen Sie die korrigierte Zeile. Steuerzeichen wie zum Beispiel Tabulatoren dürfen in einer Zeile nicht verwendet werden (und kommen auch in den abgedruckten Listings nicht vor).

Haben Sie so das gesamte Programm oder auch nur »verdächtige« Teile davon durchgearbeitet, so speichert ein letzter Druck auf die Return-Taste das korrigierte Listing. SCheck wird beendet und das fehlerfreie Programmlisting steht zu Ihrer Verfügung.

SCheck erlaubt aber noch einen zweiten Weg, das Listing korrekt einzutippen. Dabei geben Sie die Prüfsumme gleich bei der Eingabe ein. Die eckigen Klammern lassen Sie aber weg. Hinter der dreistelligen Prüfsumme fügen Sie ein Leerzeichen ein, und danach den normalen Programmtext dieser Zeile. Wenn Sie den Text so aufbereitet vorliegen haben, können Sie die automatische Überprüfung mit SCheck nutzen. Rufen Sie dazu die Option »Mit Prüfsumme« auf.

SCheck verwandelt das Listing nun in eine Quelldatei für den jeweiligen Compiler, indem die drei Prüfsummenstellen und das Leerzeichen vom Rest der Zeile abgetrennt werden. Aus der so entstandenen Programmzeile berechnet SCheck die Prüfsumme und vergleicht sie mit der zuvor abgetrennten Summe, die Sie einge-

Der Arbeitsbildschirm von SCheck 2.0b in Aktion



geben haben. Falls ein Unterschied auftritt, haben Sie Gelegenheit, die fehlerhafte Zeile manuell zu korrigieren.

Nach einem solchen Durchlauf von SCHECK steht wieder das fehlerfreie Programm, allerdings diesmal ohne Prüfsumme, auf der Diskette. Auch hier gilt, daß Sie nicht mit dem Original arbeiten sollten, sondern nur mit einer Kopie, um im Zweifelsfall nochmal von vorne beginnen zu können.

SCHECK verfügt aber noch über zwei weitere Optionen. Die von Fall zu Fall abgedruckten hexadezimalen Listings können Sie mit SCHECK im Binär-Modus direkt in lauffähige Programme umwandeln. Dazu müssen Sie das Hexdump-Listing mit einem ganz normalen Editor (zum Beispiel Turbo Pascal, Wordstar oder Edlin) als Textdatei eingeben. Anschließend überprüfen Sie mit SCHECK zunächst die Korrektheit der abgetippten Daten wie bei jedem anderen Listing. Befindet

sich das Programm korrekt auf Diskette, so wird die Option »Hexdump« aufgerufen. SCHECK wird nun dazu veranlaßt, eine Hexdump-Textdatei in ein ablauffähiges Programm umzuwandeln. SCHECK erzeugt in jedem Fall eine Datei mit der Namensweiterung »bin«. Um daraus eine lauffähige Datei zu machen, müssen Sie sie mit dem Rename-Kommando von MS-DOS umbenennen:

```
rename <name>.bin <name>.com
```

Falls eine Exe-Datei erzeugt werden soll (bitte entnehmen Sie dies der jeweiligen Programmbeschreibung) lautet der entsprechende Befehl:

```
rename <name>.bin <name>.exe
```

Sie sehen, mit SCHECK 2.0b ist das korrekte Abtippen von Programmlistings kein bißchen komplizierter geworden – aber durch einen

verbesserten Prüfsummen-Algorithmus eben noch um einiges sicherer.

(ev/hg)

SCHECK mit SBasic compilieren

Um die Eingabehilfe SCHECK mit dem SBasic-Compiler aus dieser Ausgabe des DOS Extra compilieren zu können, müssen Sie lediglich aus Programmzeile 10070 die Zeichenfolge »key off: « löschen und die Zeilen 10720, 11240 sowie 11110 bis 11170 vollständig entfernen, so daß damit die nicht unbedingt notwendige Fehlerbehandlung durch den On-Error-Befehl, den SBasic nicht verarbeiten kann, entfällt.

Name	SCHECK.bas
Sprache	GW-Basic, SBasic
Grafikkarte	jede
Besonderheiten	keine

```
[SS5] 10000 *****
[SS5] 10000 *****
[F29] 10010 * SCHECK - Prüfsummengenerator für DOS-Listings *
[ML4] 10020 *
[807] 10030 *
[AC7] 10040 *
[XS5] 10050 *
[4U0] 10060 *
[SC5] 10070 KEY OFF: DEFINT A-Z: COLOR 7,0,1
[5G8] 10080 T = 1: TEMP = 2: F = 3'EIn-/Ausgabekanäle festlegen
[332] 10090 FALSE = 0
[902] 10100 TRUE = -1
[EY7] 10110 ZEICHENS = "0123456789ABCDEFGHIJKLMNPRSTUVWXY Z"
[9H2] 10120 LAUF = FALSE
[6B2] 10130 BIN = FALSE
[851] 10140 CLS
[9S1] 10150 PRINT
[K07] 10160 PRINT CHR$(201); STRING$(78, 205); CHR$(187);
[DHC] 10170 PRINT CHR$(186); STRING$(26, 32); "**** S
Check 2.0b *****; STRING$(27, 32); CHR$(186);
[Z7G] 10180 PRINT CHR$(186); " Basic-Version von Markus
Zietlow (c) 1989 DOS International
"; CHR$(186);
[C5C] 10190 PRINT CHR$(204); "
"; CHR$(185);
[DR3] 10200 FOR ZAEHLER = 1 TO 5
[Zy6] 10210 PRINT CHR$(186); STRING$(78, 32); CHR$(186);
[CR2] 10220 NEXT ZAEHLER
[84C] 10230 PRINT CHR$(204); "
"; CHR$(185);
[00D] 10240 PRINT CHR$(186); " CHECK: ZEILE:
DATEI:
"; CHR$(186);
[P07] 10250 PRINT CHR$(200); STRING$(78, 205); CHR$(188);
[2D2] 10260 LOCATE 20, 1
[B17] 10270 PRINT CHR$(218); STRING$(78, 196); CHR$(191);
[007] 10280 PRINT CHR$(179); STRING$(78, 32); CHR$(179);
[C27] 10290 PRINT CHR$(192); STRING$(78, 196); CHR$(217);
[4V1] 10300 COLOR 7
[9F1] 10310 Y = 7
[9V6] 10320 LOCATE 7, 31: PRINT "SCHECK Version 1.x"
```

```
[PR6] 10330 LOCATE 8, 31: PRINT "SCHECK Version 2.0"
[SSG] 10340 LOCATE 21, 5: PRINT "Bitte wählen Sie mit d
en Pfeiltasten aus und bestätigen durch <RETURN>
!";
[2PD] 10350 COLOR 14, 1: FOR Q = 30 TO 50: LOCATE Y, Q:
PRINT CHR$(SCREEN(Y, Q)): NEXT Q: COLOR 7, 0
[UP6] 10360 TASTE$ = INKEY$: IF TASTE$ = "" THEN 10360
[P96] 10370 IF TASTE$ <> CHR$(0) + CHR$(72) THEN 10400
[4ND] 10380 IF Y <> 7 THEN FOR Q = 30 TO 50: LOCATE Y,
Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y - 1
[532] 10390 GOTO 10350
[1A6] 10400 IF TASTE$ <> CHR$(0) + CHR$(80) THEN 10430
[1ND] 10410 IF Y <> 8 THEN FOR Q = 30 TO 50: LOCATE Y,
Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y + 1
[232] 10420 GOTO 10350
[F35] 10430 IF TASTE$ <> CHR$(13) THEN 10360
[8N2] 10440 FOR Q = 7 TO 8
[U16] 10450 LOCATE Q, 30: PRINT STRING$(21, 32)
[KN1] 10460 NEXT Q
[8Y2] 10470 VERSION = Y - 6
[GV1] 10480 COLOR 7
[NF1] 10490 Y = 7
[MA5] 10500 LOCATE 7, 36: PRINT "Standard"
[4A6] 10510 LOCATE 8, 34: PRINT "Mit Prüfsumme"
[C65] 10520 LOCATE 9, 37: PRINT "Hexdump"
[ZSG] 10530 LOCATE 21, 5: PRINT "Bitte wählen Sie mit d
en Pfeiltasten aus und bestätigen durch <RETURN>
!";
[9PD] 10540 COLOR 14, 1: FOR Q = 30 TO 50: LOCATE Y, Q:
PRINT CHR$(SCREEN(Y, Q)): NEXT Q: COLOR 7, 0
[3R6] 10550 TASTE$ = INKEY$: IF TASTE$ = "" THEN 10550
[4B6] 10560 IF TASTE$ <> CHR$(0) + CHR$(72) THEN 10590
[BND] 10570 IF Y <> 7 THEN FOR Q = 30 TO 50: LOCATE Y,
Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y - 1
[E32] 10580 GOTO 10540
[GA6] 10590 IF TASTE$ <> CHR$(0) + CHR$(80) THEN 10620
[CND] 10600 IF Y <> 9 THEN FOR Q = 30 TO 50: LOCATE Y,
Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y + 1
[B32] 10610 GOTO 10540
[P35] 10620 IF TASTE$ <> CHR$(13) THEN 10550
[GN2] 10630 FOR Q = 7 TO 9
[126] 10640 LOCATE Q, 30: PRINT STRING$(21, 32)
[TN1] 10650 NEXT Q
[WG4] 10660 IF Y = 8 THEN LAUF = TRUE
[NA4] 10670 IF Y = 9 THEN BIN = TRUE
[RU5] 10680 LOCATE 12, 54: PRINT STRING$(25, 32)
[C9F] 10690 GOSUB 13150: LOCATE 21, 12: PRINT "Geben Si
e bitte den Dateinamen der gewünschten Datei an !
";
[9P8] 10700 Y = 12: X = 54: LAENGE = 24: MENGE$ = "alle
": GOSUB 10890
[N6D] 10710 IF (TASTE$ = CHR$(27)) OR (Q$ = "") THEN LO
CATE 12, 54: PRINT STRING$(25, 32): GOTO 10680
[LM3] 10720 ON ERROR GOTO 11160
[C74] 10730 OPEN Q$ FOR INPUT AS #T
[732] 10740 GOTO 11230
```

Listing. »SCHECK.bas« für GW-Basic


```

[302] 10750 COLOR 12
[2A6] 10760 LOCATE 8, 30: PRINT "D A T E I F E H L E R"
[VV1] 10770 COLOR 7
[R9C] 10780 GOSUB 13150: LOCATE 21, 23: PRINT "Drücken
Sie bitte irgendeine Taste !";
[B83] 10790 WHILE INKEY$ = ""
[GX2] 10800 PRINT CHR$(7);
[CE1] 10810 WEND
[6U5] 10820 LOCATE 8, 30: PRINT STRING$(21, 32)
[842] 10830 GOTO 10680
[2V0] 10840 '
[184] 10850 '*****
[UD6] 10860 '* Unterprogramm zur Dateneingabe *
[384] 10870 '*****
[6V0] 10880 '
[GG1] 10890 QS = ""
[EF2] 10900 LOCATE Y, X
[FS3] 10910 PRINT QS; CHR$(219);
[GR6] 10920 TASTE$ = INKEY$: IF TASTE$ = "" THEN 10920
[ZK5] 10930 IF (TASTE$ <> CHR$(8)) THEN 10990
[224] 10940 IF LEN(QS) = 0 THEN 10920
[X44] 10950 QS = LEFT$(QS, LEN(QS) - 1)
[6N2] 10960 LOCATE Y, X
[V44] 10970 PRINT QS; CHR$(219); " ";
[8A2] 10980 GOTO 10920
[T98] 10990 IF TASTE$ = CHR$(13) THEN LOCATE Y, X: PRIN
T QS; " ";
[639] 11000 IF (TASTE$ = CHR$(13)) OR (TASTE$ = CHR$(27
)) THEN RETURN
[C55] 11010 IF (ASC(TASTE$) < 32) THEN 10920
[NX4] 11020 IF MENGE$ <> "zahl" THEN 11050
[ZC8] 11030 IF (ASC(TASTE$) < 48) OR (ASC(TASTE$) > 5
7) THEN 10920
[892] 11040 GOTO 11070
[PU4] 11050 IF MENGE$ = "alle" THEN 11070
[E86] 11060 IF INSTR(MENGE$, TASTE$) = 0 THEN 10920
[YF6] 11070 IF LEN(QS) < LAENGE THEN QS = QS + TASTE$
[T22] 11080 GOTO 10990
[G88] 11090 Y = 10: X = 65: LAENGE = 5: MENGE$ = "zahl"
: GOSUB 10890
[BV5] 11100 IF (TASTE$ = CHR$(27)) THEN RETURN
[6U0] 11110 '
[NK2] 11120 '*****
[TR3] 11130 '* Fehler-Routine *
[RK2] 11140 '*****
[AU0] 11150 '
[KN1] 11160 CLOSE
[ZG2] 11170 RESUME 10750
[DU0] 11180 '
[8G2] 11190 '*****
[TH3] 11200 '* Hauptprogramm *
[4G2] 11210 '*****
[BU0] 11220 '
[KP3] 11230 COLOR 7: GOSUB 13150
[N43] 11240 ON ERROR GOTO 0
[K62] 11250 DATEI$ = QS
[DF7] 11260 GROSS$ = DATEI$: GOSUB 13040: DATEI$ = GROS
S$
[3W3] 11270 I = INSTR(DATEI$, ".")
[6K7] 11280 IF I = 0 THEN DATEI$ = DATEI$ + ".": I = LE
N(DATEI$)
[U64] 11290 EXT$ = MID$(DATEI$, I, 4)
[8T4] 11300 DATEI$ = LEFT$(DATEI$, I - 1)
[HN5] 11310 LOCATE 12, 54: PRINT DATEI$: EXT$
[HB4] 11320 TEMPNAME$ = DATEI$ + ".$$$"
[5E6] 11330 OPEN TEMPNAME$ FOR OUTPUT AS #TEMP
[CZ7] 11340 IF BIN THEN OPEN DATEI$ + ".BIN" FOR OUTPUT
AS #F
[CP7] 11350 IF VERSION = 2 THEN NUMBER = 1 ELSE NUMBER
= 0
[KYC] 11360 IF BIN THEN GOSUB 13150: LOCATE 21, 25: PRI
NT ">>> Binärdatei wird erzeugt <<<"
[TE3] 11370 WHILE NOT EOF(T)
[X24] 11380 LINE INPUT #T, ZEILE$
[LDB] 11385 WHILE RIGHT$(ZEILE$,1)="" : ZEILE$=LEFT$(
ZEILE$,LEN(ZEILE$)-1) : WEND
[333] 11390 WEITER = FALSE
[WT3] 11400 IF WEITER THEN 11500
[WN6] 11410 GOSUB 11820 'Gelesene Zeile anzeigen
[TS4] 11420 IF BIN THEN WEITER = TRUE
[ZT3] 11430 IF WEITER THEN 11500
[AM6] 11440 CH$ = INKEY$: IF CH$ = "" THEN GOTO 11440
[3G3] 11450 ERLAUBT = FALSE
[ZH8] 11460 IF CH$ = CHR$(13) THEN ERLAUBT = TRUE: WE
ITER = TRUE
[N1C] 11470 IF (CH$ = "e") OR (CH$ = "E") THEN ERLAUB
T = TRUE: GOSUB 12060 'Zeile bearbeiten
[8ME] 11480 IF (CH$ = "q") OR (CH$ = "Q") THEN IF NOT
LAUF THEN GOSUB 12420: IF SCHLUSS THEN GOTO 1163
0 'Quit
[F57] 11490 IF ERLAUBT THEN GOTO 11400 ELSE GOTO 1144
0
[8Y3] 11500 PRINT #TEMP, ZEILE$
[DU3] 11510 IF NOT BIN THEN 11610
[D99] 11520 WHILE LEFT$(ZEILE$, 1) = " ": ZEILE$ = MI
D$(ZEILE$, 2): WEND
[364] 11530 IF ZEILE$ = "" THEN 11610

```

```

[K75] 11540 B = VAL("&H" + LEFT$(ZEILE$, 2))
[TSH] 11550 IF (B < 0) OR (B > 255) THEN GOSUB 13150:
LOCATE 21, 22: PRINT "+++++ ABBRUCH WEGEN DATENF
EHLER +++++": LOCATE 25, 1: END
[HN1] 11560 BY = B
[TK3] 11570 PRINT #F, CHR$(BY);
[YA4] 11580 ZEILE$ = MID$(ZEILE$, 3)
[M99] 11590 WHILE LEFT$(ZEILE$, 1) = " ": ZEILE$ = MI
D$(ZEILE$, 2): WEND
[T92] 11600 GOTO 11530
[DY3] 11610 NUMBER = NUMBER + 1
[8E1] 11620 WEND
[S7B] 11630 WHILE NOT EOF(T): LINE INPUT #T, ZEILE$: PR
INT #TEMP, ZEILE$: WEND
[4P1] 11640 CLOSE
[8U5] 11650 OPEN DATEI$ + EXT$ FOR INPUT AS #1
[X06] 11660 OPEN DATEI$ + ".BAK" FOR OUTPUT AS #2
[XGB] 11670 WHILE NOT EOF(1): LINE INPUT #1, ZEILE$: PR
INT #2, ZEILE$: WEND: CLOSE
[6D5] 11680 OPEN TEMPNAME$ FOR INPUT AS #1
[576] 11690 OPEN DATEI$ + EXT$ FOR OUTPUT AS #2
[UGB] 11700 WHILE NOT EOF(1): LINE INPUT #1, ZEILE$: PR
INT #2, ZEILE$: WEND: CLOSE
[713] 11710 KILL TEMPNAME$
[GTN] 11720 GOSUB 12560: LOCATE 12, 9: PRINT STRING$(11
, 32): LOCATE 12, 25: PRINT STRING$(21, 32): LOCA
TE 12, 53: PRINT STRING$(27, 32) 'Statuszeile lösc
hen
[494] 11730 IF BIN THEN EXT$ = ".BIN"
[N4H] 11740 GOSUB 13150: LOCATE 21, (80 - (31 + LEN(DAT
EI$ + EXT$))) / 2: PRINT "Die Datei "; DATEI$ +
EXT$; " wurde gespeichert."
[CE2] 11750 LOCATE 25, 1
[H51] 11760 END
[4V0] 11770 '
[TH3] 11780 '*****
[B65] 11790 '* Gelesene Zeile anzeigen *
[NH3] 11800 '*****
[2V0] 11810 '
[BBB] 11820 IF LAUF THEN CHECK$ = "[" + LEFT$(ZEILE$, 3
) + "]" : ZEILE$ = MID$(ZEILE$, 5)
[MDN] 11830 FOR Q = 1 TO LEN(CHECK$): IF ASC(MID$(CHECK
$, Q, 1)) > 96 AND ASC(MID$(CHECK$, Q, 1)) < 123
THEN MID$(CHECK$, Q, 1) = CHR$(ASC(MID$(CHECK$, Q
, 1)) - 32)
[3P1] 11840 NEXT Q
[B57] 11850 LINKS = 2: OBEN = 6: RECHTS = 79: UNTEN = 1
0
[CX4] 11860 LOCATE 12, 26: PRINT NUMBER
[Z45] 11870 GOSUB 12560 'Fenster löschen
[WX8] 11880 GOSUB 12660 'Zeile innerhalb des Fensters an
zeigen
[366] 11890 GOSUB 12800 'Check-Summe errechnen
[0M2] 11900 LOCATE 12, 10
[HB2] 11910 PRINT PSS
[HM7] 11920 IF BIN THEN GOTO 12000 'Binärdatei erzeugen
[N8C] 11930 IF LAUF THEN GOTO 11970 'Übereinstimmung der
Prüfsummen feststellen
[X74] 11940 GOSUB 13150: LOCATE 21, 4
[LKD] 11950 PRINT "[E] Editieren [RETURN] Weiter
[Q] Quit [Ctrl-Break] Abbrechen"
[152] 11960 GOTO 11990
[ATP] 11970 GOSUB 13150: IF PSS = CHECK$ THEN LOCATE 21
, (80 - (22 + LEN(STR$(NUMBER)))) / 2: PRINT "Prü
fsumme OK für Zeile"; NUMBER: WEITER = TRUE: GOTO
12000
[1NH] 11980 LOCATE 21, 4: PRINT CHR$(7); "Prüfsummenfeh
ler! [E] Editieren [RETURN] Weiter [Ctrl-Break]
Abbrechen"
[XM2] 11990 LOCATE 12, 15
[KV1] 12000 RETURN
[5U0] 12010 '
[MK2] 12020 '*****
[4H3] 12030 '* Zeile eingeben *
[PK2] 12040 '*****
[9U0] 12050 '
[FVD] 12060 GOSUB 13150: LOCATE 21, 17: PRINT "Ändern S
ie die Programmzeile ! [RETURN] Ende"
[245] 12070 GOSUB 12560 'Fenster löschen
[7B5] 12080 GOSUB 12660 'Zeile darstellen
[FL3] 12090 LOCATE OBEN, LINKS
[7G3] 12100 X = 1: PRINT "■";
[8E2] 12110 CH$ = INKEY$
[FE6] 12120 IF LEFT$(CH$, 1) <> CHR$(0) THEN GOTO 12220
[4N3] 12130 CH$ = RIGHT$(CH$, 1)
[4NE] 12140 IF CH$ = "M" THEN IF X > LEN(ZEILE$) THEN
PRINT CHR$(7); : GOTO 12110 ELSE GOSUB 12340: X
= X + 1
[KZC] 12150 IF CH$ = "K" THEN IF X = 1 THEN PRINT CHR
$(7); : GOTO 12110 ELSE GOSUB 12340: X = X - 1
[YZC] 12160 IF CH$ = "H" THEN IF X < 79 THEN PRINT CH
R$(7); : GOTO 12110 ELSE GOSUB 12340: X = X - 78
[YCF] 12170 IF CH$ = "P" THEN IF X + 77 > LEN(ZEILE$)
THEN PRINT CHR$(7); : GOTO 12110 ELSE GOSUB 1234
0: X = X + 78
[TV7] 12180 IF CH$ = "G" THEN IF X <> 1 THEN GOSUB 12
340: X = 1

```

Listing. »Scheck.bas« für GW-Basic


```

[41C] 12190 IF CH$ = "O" THEN IF X <> LEN(ZEILE$) + 1
      THEN GOSUB 12340: X = LEN(ZEILE$) + 1
[8PM] 12200 IF CH$ = "S" THEN IF X <= LEN(ZEILE$) THE
      N ZEILE$ = LEFT$(ZEILE$, X - 1) + MID$(ZEILE$, X
      + 1): GOSUB 12660 ELSE PRINT CHR$(7); : GOTO 1211
      0
[G92] 12210 GOTO 12240
[XXN] 12220 IF CH$ = CHR$(8) THEN IF X > 1 THEN ZEILE$
      = LEFT$(ZEILE$, X - 2) + MID$(ZEILE$, X): GOSUB 1
      2660: GOSUB 12340: X = X - 1 ELSE PRINT CHR$(7);
      : GOTO 12110
[LZN] 12230 IF CH$ >= " " THEN IF LEN(ZEILE$) = 255 THE
      N PRINT CHR$(7); : GOTO 12110 ELSE ZEILE$ = LEFT$
      (ZEILE$, X - 1) + CH$ + MID$(ZEILE$, X): GOSUB 12
      660: X = X + 1
[7YB] 12240 LOCATE OBEN + INT((X - 1) / 78), LINKS - 1
      + X - INT((X - 1) / 78) * 78: PRINT "█":
[LK5] 12250 IF CH$ <> CHR$(13) THEN GOTO 12110
[648] 12260 IF LAUF THEN ZEILE$ = MID$(CHECK$, 2, 3) +
      " " + ZEILE$
[XL6] 12270 GOSUB 12340: GOSUB 12660 'Zeile anzeigen
[2W1] 12280 RETURN
[NU0] 12290 '
[CP3] 12300 '*****
[2H5] 12310 '* Zeichen wiederherstellen *
[EP3] 12320 '*****
[LU0] 12330 '
[UM9] 12340 LOCATE OBEN + INT((X - 1) / 78), LINKS - 1
      + X - INT((X - 1) / 78) * 78
[YXA] 12350 IF MID$(ZEILE$, X, 1) = "" THEN PRINT " " E
      LSE PRINT MID$(ZEILE$, X, 1)
[4W1] 12360 RETURN
[RU0] 12370 '
[0A2] 12380 '*****
[3X2] 12390 '* Quit (Ende) *
[W92] 12400 '*****
[NU0] 12410 '
[U63] 12420 SCHLUSS = FALSE
[KYB] 12430 SICHER$ = "": FOR Q = 2 TO 79: SICHER$ = SI
      CHER$ + CHR$(SCREEN(21, Q)): NEXT Q
[7LD] 12440 GOSUB 13150: LOCATE 21, 16: PRINT "Drücken
      Sie bitte [J], um die Arbeit zu beenden !"
[GA6] 12450 CH$ = INKEY$: IF CH$ = "" THEN GOTO 12450
[R3B] 12460 IF (CH$ = "j") OR (CH$ = "J") THEN SCHLUSS
      = TRUE: PRINT #TEMP, ZEILE$
[BA2] 12470 GOSUB 13150
[P78] 12480 IF NOT SCHLUSS THEN LOCATE 21, 2: PRINT SIC
      HER$
[BW1] 12490 RETURN
[651] 12500 END
[TU0] 12510 '
[6R2] 12520 '*****
[U24] 12530 '* Fenster löschen *
[8R2] 12540 '*****
[XU0] 12550 '
[044] 12560 FOR Q = OBEN TO UNTEN
[Z53] 12570 LOCATE Q, LINKS
[8B6] 12580 PRINT STRING$(RECHTS - LINKS + 1, 32)
[ZN1] 12590 NEXT Q
[AW1] 12600 RETURN
[XU0] 12610 '
[265] 12620 '*****
[7W7] 12630 '* Inhalt von "zeile$" im Fenster anzeigen
      *
[465] 12640 '*****

```

```

[1V0] 12650 '
[SR2] 12660 F.X = 0: F.Y = 0
[CC5] 12670 FOR ZAEHLER = 1 TO LEN(ZEILE$)
[F85] 12680 LOCATE OBEN + F.Y, LINKS + F.X
[UH5] 12690 PRINT MID$(ZEILE$, ZAEHLER, 1);
[SK2] 12700 F.X = F.X + 1
[597] 12710 IF F.X >= RECHTS - 1 THEN F.X = 0: F.Y =
      F.Y + 1
[6S2] 12720 NEXT ZAEHLER
[RX6] 12730 LOCATE OBEN + F.Y, LINKS + F.X: PRINT " ";
[KW1] 12740 RETURN
[5V0] 12750 '
[AH4] 12760 '*****
[ET6] 12770 '* Zeile auf Check-Summe überprüfen *
[CH4] 12780 '*****
[9V0] 12790 '
[F42] 12800 P = 0: A = 1
[8SB] 12810 WHILE (MID$(ZEILE$, A, 1) = " ") OR (MID$(Z
      EILE$, A, 1) = CHR$(9)): A = A + 1: WEND
[374] 12820 FOR I = A TO LEN(ZEILE$)
[4CX] 12830 IF MID$(ZEILE$, I, 1) >= " " THEN GROSS$
      = MID$(ZEILE$, I, 1): IF VERSION = 1 THEN P = ((I
      - A + 1) AND 7) * ASC(GROSS$) + P: GOTO 12920 EL
      SE GOSUB 13040: P = (((I - A + 1) AND 3) + 1) * A
      SC(GROSS$) + P: GOTO 12920
[S22] 12840 COLOR 12
[5N6] 12850 LOCATE 8, 30: PRINT "D A T E I F E H L E
      R"
[022] 12860 COLOR 7
[K2C] 12870 LOCATE 21, 13: PRINT " Drücken S
      ie bitte irgendeine Taste ! ";
[CC3] 12880 WHILE INKEY$ = ""
[D93] 12890 PRINT CHR$(7);
[5L1] 12900 WEND
[R16] 12910 LOCATE 8, 30: PRINT STRING$(21, 32)
[0P1] 12920 NEXT I
[KF2] 12930 PSS$ = "[ ]"
[AM2] 12940 FOR I = 2 TO 4
[5T7] 12950 MID$(PSS$, I, 1) = MID$(ZEICHEN$, (P AND 3
      1) + 1, 1)
[H22] 12960 P = FIX(P / 32)
[5P1] 12970 NEXT I
[YW1] 12980 RETURN
[HV0] 12990 '
[ZX3] 13000 '*****
[K06] 13010 '* In Großbuchstaben umwandeln *
[934] 13020 '*****
[AU0] 13030 '
[9ZN] 13040 FOR Q = 1 TO LEN(GROSS$): IF ASC(MID$(GROSS
      $, Q, 1)) > 96 AND ASC(MID$(GROSS$, Q, 1)) < 123
      THEN MID$(GROSS$, Q, 1) = CHR$(ASC(MID$(GROSS$, Q
      , 1)) - 32)
[8T8] 13050 IF MID$(GROSS$, Q, 1) = "ä" THEN MID$(GROSS
      $, Q, 1) = "Ä"
[NU8] 13060 IF MID$(GROSS$, Q, 1) = "ö" THEN MID$(GROSS
      $, Q, 1) = "Ö"
[UU8] 13070 IF MID$(GROSS$, Q, 1) = "ü" THEN MID$(GROSS
      $, Q, 1) = "Ü"
[DN1] 13080 NEXT Q
[YV1] 13090 RETURN
[BU0] 13100 '
[Y22] 13110 '*****
[9N4] 13120 '* Meldezeile löschen *
[003] 13130 '*****
[FU0] 13140 '
[ZT5] 13150 LOCATE 21, 2: PRINT STRING$(78, 32)
[ZV1] 13160 RETURN

```

Listing. »SCheck.bas« für GW-Basic

Lieben Sie es,...

abends über Ihrer Buchführung zu brüten?

Lästige Routinearbeiten nimmt Ihnen unsere professionelle Büro-Software ab:

Auftrag – Angebot-Lager-Faktura

Angebot, Auftrag, Rechnung, Lieferschein,
Lager, Mahnung, Bestellwesen,
Kunden, Lieferanten, Serienbriefe usw.

499,-

Finanzbuchhaltung

Bilanz, SS, BWA, Journal, DATEV-Kontenrahmen,
GuV, Ust.-Vorabm., Kreditoren, Debitoren usw.

499,-

Lohnbuchhaltung

KK- und Finanzamtliste, Lohnabrechnung,
Überweisungsträger usw.

499,-

Branchenlösungen

■ Bauhaupt- und -nebgewerbe, Architekten
■ Versicherungsagenturen ■ Autovermietungen

Preis auf Anfrage

Alle Programme

- extrem einfach in der Bedienung
- menügesteuert mit Fenstertechnik
- individuell anpaßbar, mehrplatzfähig
- Ihr Netzwerk-Spezialist

Wir schicken Ihnen gern kostenlose eine Demo-Disk und
informieren Sie unverbindlich über unsere Büro-Software:

SOFTWARE & HARDWARE, Ingenieurbüro Bitter,
Telefon: 04 21/21 05 11, Hauptmann-Böse-Weg 2-4, 2800 Bremen 1
■ DELA-EDV, Köln 1, Telefon: 02 21/51 70 84
■ Büttner-EDV, Bielefeld, Tel.: 05 21/49 39 94

Der Befehlssatz von GW-Basic im Überblick

Wenn Sie es leid sind, erst umständlich im Handbuch zu blättern, um einen ganz bestimmten Befehl oder die Syntax zu einem Befehl zu finden, dann ist die große Basic-Referenztafel genau das richtige für Sie. Einige der Parameter oder gar komplette Befehle finden Sie wahrscheinlich nicht einmal in Ihrem Handbuch.

Die GW-Basic-Referenz listet alle Befehle, Funktionen und Systemvariablen der aktuellen GW-Basic-Version 3.22 auf. Ist ein Parameter nicht zwingend erforderlich und wird er vom Programmierer im Befehl nicht eingesetzt, so nimmt der GW-Ba-

sic-Interpreter automatisch den Wert 0 an. In Fällen, in denen eine andere Zahl als Standardwert benutzt wird (zum Beispiel bei der Schrittweite einer For-Next-Schleife), wird dieser Umstand gesondert vermerkt. Die Erklärungen zu den einzelnen

Befehlen sind aus Platzgründen sehr knapp gehalten. Zum Schluß noch ein paar Regeln zum Umgang mit der Referenztafel:

- GW-Basic-Befehle sind grundsätzlich groß geschrieben.
- Parameter (Variablen und Werte) sind grundsätzlich klein geschrieben und in spitze Klammern (»<« und »>«) eingeschlossen, um Verwechslungen vorzubeugen.
- Ausdrücke in eckigen Klammern (»[« und »]«) sind wahlweise und können weggelassen werden.
- Ist ein Ausdruck in eckige Klammern eingeschlossen, so befindet sich auch der zugehörige Text in der Erklärung innerhalb eckiger Klammern.

(ma/tr)

Befehl und Parameter	Bedeutung
<Argument1> AND <Argument2>	Verknüpfe Argument1 mit Argument2 über die logische Und-Funktion
AUTO <Zeile1> [, <Zeile2>]	Erzeuge automatisch Zeilennummern ab Zeile1 [mit Schrittweite Zeile2]
BEEP	Erzeuge Piepston, der etwa eine Viertelsekunde lang anhält (entspricht »chr\$(7)«)
BLOAD <Dateiname> [, <Offset>]	Lese Binärdatei [ab Adresse »Offset«] von Diskette oder Festplatte in den Arbeitsspeicher
BSAVE " <Dateiname> ", <Offset> [, <Länge>]	Schreibe Binärdatei ab Adresse »Offset« mit einer vorgegebenen Länge an Bytes auf Diskette oder Festplatte
CALL <Variable> [, <Parameter>]	Aufruf eines Maschinenprogramms ab einer in der Variablen enthaltenen Offset-Adresse des Def-Seg-Segments [und Übergabe der Parameter als Pointer auf dem System-Stack]
CHAIN [MERGE] " <Dateiname> " [, <Zeile>] [, ALL] [, DELETE <Bereich>]	Lade Programm von Diskette oder Festplatte und führe es [ab Programmzeile »Zeile«] aus. »ALL« stellt alle Variablen dem nachgeladenen Programm zur Verfügung (siehe auch Common-Befehl). »MERGE« löscht das alte Programm nicht, sondern hängt das neue lediglich an die letzte Zeile »Datei« muß im ASCII-Format vorliegen. Die DELETE-Option löscht vor dem Laden die Zeilennummern »Bereich« aus dem aufrufenden Programm (nur in Verbindung mit »MERGE«)
CHDIR " <Pfad> "	Ändere das aktuelle Verzeichnis auf der Diskette oder Festplatte zu »Pfad«
CIRCLE [STEP] (<x> [, <y>], <Radius> [, <Farbe>] [, <Winkel1> [, <Winkel2>] [, <Verhältnis>]	Zeichne Ellipse, deren Mittelpunkt auf der Koordinate (x,y) liegt, mit dem Radius »Radius« [in der Farbe »Farbe«] [von Winkel »Winkel1« bis »Winkel2« im Gradmaß] [und dem Achsenverhältnis »Verhältnis«]. Ist »STEP« angegeben, sind »x« und »y« relativ zum zuletzt gezeichneten Grafikpunkt
CLEAR [<Datensegment>] [, <Stackgröße>]	Lösche alle Variablenwerte und reserviere »Datensegment« Bytes als Datensegment (Default 65535) sowie »Stackgröße« Bytes für den Stapel (Default 512)
CLOSE [[#] <Kanal1> [, [#] <Kanal2> [, ...]]	Schließe alle beziehungsweise die angegebenen Dateikanäle
CLS	Lösche aktuelle Bildschirmseite und setze relative Grafikkoordinaten auf 160/100 beziehungsweise 320/100
COLOR [<Vordergrund>] [, <Hintergrund>] [, <Rahmen>]	Textmodus: Wähle für Textzeichen Farbe »Vordergrund«, für den Hintergrund Farbe »Hintergrund« und für den Rahmen Farbe »Rahmen«

Befehl und Parameter	Bedeutung
COLOR [<Hintergrund>] [, <Palette>]	Grafikmodus: Wähle für die Grafik die Farbpalette »Palette« (0 oder 1) und für den Hintergrund die Farbe »Hintergrund«
COM(<n>) [ON] [OFF] [STOP]	Sperre (OFF) oder erlaube (ON) Unterbrechungsanforderungen der seriellen Ports »n« durch den On-Com-Befehl. »STOP« unterbricht die Überwachung und merkt sich Anforderungen bis zum nächsten »COM ON«
COMMON <Variablen>	Übergebe die Variablen »Variablen« an ein über den Chain-Befehl aufgerufenes Programm (siehe auch ALL-Option des Chain-Befehls)
CONT	Setze ein unterbrochenes Programm fort
DATA <Konstanten>	Datenfeld für den Read-Befehl, bestehend aus den Daten »Konstanten«, durch Kommata getrennt
DATE\$ = " <Datumstring> "	Setze das Datum »Datum« in einem beliebigen nationalen Format (»dd-mm-yy«)
DEF FN <Funktion> [(<Variablen>)] = <Ausdruck>	Definiere die Funktion »Funktion« mit der mathematischen Anweisung »Ausdruck«, abhängig von den Variablen »Variablen«
DEFDBL <Variable1> [, <Variable2>] [, <Variable3> [, <Variable4>]]...	Definiere die angegebenen Variablen oder Variablenbereiche als doppelgenaue Realvariablen
DEFINT <Variable1> [, <Variable2>] [, <Variable3> [, <Variable4>]]...	Definiere die angegebenen Variablen oder Variablenbereiche als Integer-Variablen
DEF SEG [= <Segment>]	Alle folgenden Offset-Speicheradressen beziehen sich auf »Segment« (verbindlich für die Befehle »BLOAD«, »BSAVE«, »CALL«, »DEFUSR«, »PEEK« und »POKE«). »DEF SEG« ohne Parameter schaltet auf das Basic-Datensegment
DEFSNG <Variable1> [, <Variable2>] [, <Variable3> [, <Variable4>]]...	Definiere die angegebenen Variablen oder Variablenbereiche als einfachgenaue Realvariablen
DEFSTR <Variable1> [, <Variable2>] [, <Variable3> [, <Variable4>]]...	Definiere die angegebenen Variablen oder Variablenbereiche als String-Variablen
DEF USR[<Nummer>] = <Offset>	Definiere Adresse »Offset« als Startadresse im aktuellen Segment für den Usr-Aufruf [mit der Nummer »Nummer«]
DELETE [<Zeile1>] [, <Zeile2>]	Lösche [Programmzeile »Zeile1«] [bis »Zeile2«]. »Delete« ohne Parameter löscht das ganze Programm
DIM <Variable> (<Index>) [, <Variable> (<Index>)]	Dimensioniere die Feldvariable »Variable« bis zum Index »Index«. Siehe auch Befehl »option base«

Befehl und Parameter	Bedeutung
DRAW <String>	Zeichne ein Objekt auf den Bildschirm (Parameter: Siehe Artikel »GW-Basic-Referenz: Grafik« in diesem Sonderheft)
EDIT <Zeile>	Liste Programmzeile »Zeile« auf dem Bildschirm und positioniere Cursor auf erstes Zeichen
END	Programmende
ENVIRON <String>	Modifiziere Betriebssystemumgebung, ähnlich dem MS-DOS-Set-Befehl. »String« hat die Form »Variable=Parameter«
ERASE <Variablen>	Lösche die Feldvariablen »Variablen«. Eine Indexangabe ist nicht nötig
ERROR <n>	Erzwinge Fehlermeldung »n« und setze »err« entsprechend, löst On-Error-Befehl aus
FIELD # <Kanal>, <Bytes> AS <Stringvariable>	Reserviere im Puffer von Kanal »Kanal« für die Variable »Stringvariable« Speicher von »Bytes« Bytes
FILES [<Dateien>]	Zeige [eine Auswahl aus dem] Verzeichnis wie der MS-DOS-Befehl »dir /w«
FOR <Variable> = <Start> TO <Ende> [STEP <Zähler>]	Schleife, in der die Laufvariable »Variable«, ausgehend vom Wert »Start« [mit einer Schrittweite von »Zähler«] bis auf »Ende« herauf- oder heruntergezählt wird
GET # <Kanal> [, <Record>]	Lese einen Datensatz [mit der Nummer »Record«] über den Dateikanal »Kanal« aus einer Random-Access-Datei
GET [STEP] (<x1>, <y1>)- (<x2>, <y2>), <Feldvariable>	Lese die Bildpunktinformationen aus dem durch die Punkte (x1,y1) und (x2,y2) festgelegten rechteckigen Bildschirmfenster in die numerische Feldvariable »Feldvariable«
GOSUB <Zeile>	Rufe ein durch »return« abgeschlossenes Unterprogramm ab Programmzeile »Zeile« auf
GOTO <Zeile>	Setze Programmausführung bei Zeile »Zeile« fort, steht immer als letzter (oder einziger) Befehl in einer Basic-Zeile
IF <Argument> THEN <Ausdruck1> [ELSE <Ausdruck2>]	Wenn der Vergleich innerhalb von »Argument« wahr ist (ungleich Null), dann führe den Befehl »Ausdruck1« aus. Falls der Vergleich unwahr (gleich Null) ist, führe die nächste Programmzeile aus [führe den Befehl »Ausdruck2« aus]. »then Zeile« ist gleichbedeutend mit »then goto Zeile«
INPUT [:] [" <Text> " <:,>] <Variable> [, <Variable>]	[Ausgabe von »Text« und] Abfrage der Werte für die Variablen »Variable« [ein Komma statt Semikolon vor der Variablen unterdrückt das Fragezeichen]. Ein Semikolon vor dem Text verhindert das Ausgeben von [Return] nach der Eingabe
INPUT # <Kanal> , <Variable> [, <Variable>]	Lese die Werte für »Variable« über den Dateikanal mit der Nummer »Kanal« ein. Ist das erste Zeichen des eingelesenen Strings ein Anführungszeichen, liest »INPUT #« bis zum nächsten Anführungszeichen. Ansonsten bestimmen ein Komma, ein [Line Feed] (chr\$(10)) oder ein [Carriage Return] (chr\$(13)) das String-Ende
IOCTL # <Kanal> , <String>	Schreibe die Steuerzeichenkette »String« über den Dateikanal »Kanal« in einen Gerätetreiber (ab GW-Basic 3.0)
KEY <Taste> , <String>	Belege die Funktionstaste »Taste« mit der Zeichenkette »String«
KEY [LIST] [ON] [OFF]	Zeige die vollständige Belegung aller Funktionstasten auf dem Bildschirm an ([LIST]) oder schalte die Funktionstastenanzeige in der unteren Bildschirmzeile an ([ON]) oder aus ([OFF])
KEY (<Taste>) [ON] [OFF] [STOP]	Sperre ([OFF]), erlaube ([ON]) oder merke ([STOP]) bis zum nächsten »on« alle Unterbrechungsanforderungen der Funktionstaste »Taste«
KILL <Datei>	Lösche Datei oder Verzeichnis »Datei« von Diskette oder Festplatte (äquivalent zum MS-DOS-Befehl »del«)
[LET] <Variable> = <Argument>	Weise der Variablen »Variable« das Ergebnis von »Argument« zu. Der Let-Befehl ist optional

Befehl und Parameter	Bedeutung
LINE [STEP] (<x1>, <y1>)- [STEP] (<x2>, <y2>) [, <Farbe>] [,B[F]] [, <Muster>]	Zeichne eine Linie von der Grafikordinate (x1,y1) zur Koordinate (x2,y2) [in der Farbe »Farbe«]. Ist »STEP« angegeben, sind die Koordinaten relativ zum zuletzt gesetzten Grafikpunkt. Mit »B« zeichnet Basic ein Rechteck mit den genannten Eckkoordinaten und füllt dieses bei Bedarf aus (»BF«). »Muster« ist eine 16-Bit-Zahl, deren gesetzte Bits das Linienmuster bestimmen
LINE INPUT [:] [" <Text> "] <Variable>	[Ausgabe von Text »Text« und] Abfrage einer Zeichenkettenzeile für die Variable »Variable«. Liest auch Anführungszeichen und Kommata
LINE INPUT # <Kanal> , <Variable>	Lese eine Zeichenkettenzeile über den Dateikanal »Kanal« in die Variable »Variable«. Als String-Ende wird ausschließlich eine [Carriage Return]-[Line Feed]-Kombination (chr\$(13)chr\$(10)) erkannt
LIST [<Zeile1>] [- <Zeile2>] [, <Datei>]	Liste Basic-Programm [von Zeile »Zeile1«] [bis »Zeile2«] auf dem Bildschirm [in Datei oder zum Gerät »Datei«]
LLIST [<Zeile1>] [- <Zeile2>]	Liste Basic-Programm [von Zeile »Zeile1«] [bis »Zeile2«] auf den Drucker
LOAD <Datei> [,R]	Lade Datei »Datei« von Diskette oder Festplatte in den Arbeitsspeicher [und starte das Programm selbstständig] Default-Erweiterung ist ».bas«
LOCATE [<Spalte>] [, <Zeile>] [, <Cursor>] [, <Zeile1>] [, <Zeile2>]]]	Positioniere den Text-Cursor auf »Spalte«, »Zeile« mit sichtbarem oder unsichtbarem Cursor (»Cursor« gleich 1 oder gleich 0). »Zeile1« und »Zeile2« sind Scan-Zeilen und bestimmen die Form des Cursor
LPRINT <String> [, <String> [:]	Gebe Text und/oder Variablen »String« auf dem Drucker aus. Ein Semikolon am Ende des Befehls verhindert das Senden von [CR/LF]
LPRINT USING <Format> ; <String> [, <String> [:]	Gebe Text und/oder Variablen »String« im Format »Format« auf den Drucker aus. Ein Semikolon am Ende des Befehls verhindert das Senden von [CR/LF]
LSET <Variable> = <String>	Lege die Daten der Zeichenkette »String« linksbündig in dem durch den Field-Befehl definierten Datenfeld der String-Variablen »Variable« ab
MERGE <Datei>	Lade Programm »Datei« von Diskette oder Festplatte und füge es in das bestehende Programm ein. Das nachgeladene Programm überschreibt gleiche Zeilennummern
MID\$ (<String1> , <Position> [, <Länge>]) = <String2>	Füge [»Länge« Zeichen der] Zeichenkette »String2« ab der Position »Position« in die Zeichenkette »String1« ein
MKDIR <Pfad>	Richte das Verzeichnis »Pfad« auf der Diskette/Festplatte neu ein
NAME <Datei1> AS <Datei2>	Benenne Datei »Datei1« in eine Datei mit dem Namen »Datei2« um
NEW	Lösche das im Arbeitsspeicher befindliche Programm, sämtliche Variablen und schalte den Trace-Modus aus
NEXT [<Variable>]	Ende der For-Next-Schleife [mit der Laufvariablen »Variable«]. Siehe »FOR«
NOT <Argument>	Invertiere den logischen Wert von »Argument«
ON COM(<n>) GOSUB <Zeile>	Rufe Unterprogramm ab Programmzeile »Zeile« auf, wenn durch die Daten auf dem seriellen Port »n« eine Unterbrechungsanforderung vorliegt (siehe »com(n)...«)
ON ERROR GOTO <Zeile>	Rufe Programmzeile »Zeile« auf, wenn ein Fehler aufgetreten ist. Fehlernummer steht in Variable »ERR«
ON <Argument> GOSUB <Zeile> [, <Zeile>]	Rufe Unterprogramm ab Programmzeile »Zeile« abhängig von »Argument« auf (»Argument«=1 bewirkt Aufruf der ersten aufgeführten Zeile, »Argument«=2 bewirkt Aufruf der zweiten Zeile und so weiter)

Befehl und Parameter	Bedeutung
ON <Argument> GOTO <Zeile> [, <Zeile>]	Setze Programmausführung abhängig von »Argument« fort (»Argument«=1 bewirkt Aufruf der ersten aufgeführten Zeile, »Argument«=2 bewirkt Aufruf der zweiten Zeile und so weiter)
ON KEY(<Taste>) GOSUB <Zeile>	Rufe jedesmal Unterprogramm ab Programmzeile »Zeile« auf, wenn die Funktionstaste »Taste« gedrückt wurde (siehe auch »key(Taste)...«)
ON PEN GOSUB <Zeile>	Rufe jedesmal Unterprogramm ab Programmzeile »Zeile« auf, wenn der Lichtgriffel aktiviert wurde (siehe auch »pen...«)
ON PLAY (<Anzahl>) GOSUB <Zeile>	Rufe jedesmal Unterprogramm ab Programmzeile »Zeile« auf, wenn in der Tonwarteschlange nur noch »Anzahl« Töne vorhanden sind (siehe auch »play...«)
ON STRIG (<Knopf>) GOSUB <Zeile>	Rufe jedesmal Unterprogramm ab Programmzeile »Zeile« auf, wenn der Joystick-Feuerknopf »Knopf« (0, 2 4 oder 6) gedrückt wurde (siehe auch »strig...«)
ON TIMER (<Sekunden>) GOSUB <Zeile>	Rufe jedesmal Unterprogramm ab Programmzeile »Zeile« auf, wenn auf der Uhr ein Zeitraum von »Sekunden« Sekunden verstrichen ist (Freigabe durch »timer on«)
OPEN <Datei> [FOR <Modus>] AS # <Kanal> [LEN=<Länge>]	Öffne Kanal »Kanal« für Datei »Datei«. »Modus« bestimmt die Art des Zugriffs: »INPUT« (Lesen), »OUTPUT« (Schreiben) oder »APPEND« (Anhängen). Ohne »FOR...« wird der Kanal zum wahlfreien Zugriff mit der Datensatzlänge »Länge« geöffnet (Default 128).
OPEN " <Modus> ", # <Kanal> , <Datei> [, <Länge>]	Öffne Kanal »Kanal« für Datei »Datei« mit der Zugriffsart Modus: »I« = sequentielles Lesen, »O« = sequentielles Schreiben, »A« = sequentielles Anhängen von Daten, »R« = wahlfreier Zugriff [mit Datensatzlänge »Länge«]
OPEN "COM<n>:[<Baudrate>] [, <Parität>] [, <Datenbits>] [, <Stopbits>] [, <weitere Parameter>]" AS # <Kanal> [LEN=<Maximum>]	Öffne Kanal »Kanal« für Datenfernübertragung mit den angegebenen Übertragungsparametern (»weitere Parameter« und »Maximum« steuern den Betrieb der seriellen Schnittstelle).
OPTION BASE <n>	Definiere »n« als kleinsten Indexwert (0 oder 1) für eine Feldvariable
<Argument1> OR <Argument2>	Verknüpfe »Argument1« mit »Argument2« über die logische Oder-Funktion
OUT <Offset> , <Byte>	Gebe Bytewert »Byte« an die Portadresse »Offset«
PAINT [STEP] (<x> , <y>) [, <Farbe>] [, <Umgebungs- farbe>] [, <Hintergrundfarbe>]	Fülle einen Bildschirmbereich ausgehend von Position (x,y) (ist »STEP« angegeben, sind die Koordinaten relativ zum zuletzt gezeichneten Punkt) mit der Farbe »Farbe« oder mit einem Muster (wenn »Farbe« ein String ist), bis die »Umgebungsfarbe« erreicht wurde
PALETTE <Farbnummer> , <Farbe>	Weise dem Farbregister »Farbnummer« die Farbe »Farbe« zu
PALETTE USING <Variable>	Weise den Farbregistern 0 bis 3 die Farbnummern aus der Integer-Feldvariablen »Variable« und den drei folgenden Feldvariablen zu
PEN [ON] [OFF] [STOP]	Erlaube ([ON]), sperre ([OFF]) oder speichere ([STOP]) Unterbrechungsanforderungen des Lichtgriffels
PLAY <String>	Spiele eine durch Zeichenkette »String« definierte Tonfolge (nähere Erklärungen: siehe entsprechenden Artikel in diesem Sonderheft)
PLAY [ON] [OFF] [STOP]	Erlaube ([ON]), sperre ([OFF]) oder speichere ([STOP]) Unterbrechungsanforderungen der Tonwarteschlange
POKE <Offset> , <Wert>	Schreibe Wert von »Wert« in Speicherzelle »Offset«
PRESET (<x> , <y>) [, <Farbe>]	Setze Grafikpunkt auf die Koordinate (x,y) in der Hintergrundfarbe [oder in der Farbe »Farbe«]

Befehl und Parameter	Bedeutung
PRINT [# <Kanal>] <Zeichen> [;]	Gebe Text und/oder Variablen »Zeichen« auf dem Bildschirm [oder dem Dateikanal »Kanal«] aus [ohne [CR/LF] am Ende der Zeile]
PRINT [# <Kanal> ,] USING <Format> ; <Variablen> [;]	Gebe Variablen »Variablen« im Format »Format« auf dem Bildschirm [oder dem Dateikanal »Kanal«] aus [ohne [CR/LF] am Ende der Zeile]
PSET [STEP] (<x> , <y>) [, <Farbe>]	Setze Grafikpunkt auf die (relative, wenn »STEP« angegeben) Koordinate (x,y) [in der Farbe »Farbe«]
PUT [#] <Kanal> [, <Record>]	Schreibe den nächsten Datensatz [mit der Nummer »Record«] aus einem Datenpuffer über den Dateikanal »Kanal«
PUT [STEP] (<x> , <y>), <Variable> [, <Form>]	Kopiere die Bildpunktinformationen aus der numerischen Feldvariable »Variable« in das Bildschirmfenster, dessen linke obere Ecke durch die (relative, wenn »STEP« angegeben) Koordinate (x,y) gebildet wird. »Form« ist die Art des bitweisen Zurschreibens: »PSET«, »PRESET«, »XOR«, »OR« oder »AND«
RANDOMIZE [<Wert>]	Lege [den Wert »Wert« als] Ausgangswert für den Zufallszahlengenerator fest. Lautet meist »randomize timer«
READ <Variable> [, <Variable>]	Lese für die Variable »Variable« die Werte aus Data-Zeilen
REM <Text>	Berücksichtige den dem Rem-Befehl nachfolgenden »Text« nicht (dient zur Programmdokumentation). Gleichbedeutend mit einem Apostroph »'«
RENUM [<Startzeile>] [, <Altzeile>] [, <Schrittweite>]	Numeriere das Programm neu [ausgehend von der neuen Programmzeile »Startzeile«] (oder 10) [ab der alten Programmzeile »Altzeile«] [mit der Schrittweite »Schrittweite«] (oder 10)
RESET	Schließe alle Diskettenkanäle und lösche den Systempuffer
RESTORE [<Zeile>]	Setze den DATA-Zeiger [auf den ersten Wert der Programmzeile »Zeile«] zurück
RESUME	Setze Programm nach einem ON-ER-ROR-GOTO-Befehl mit derjenigen Anweisung fort, die der fehlerverursachenden folgt
RESUME <Zeile>	Setze Programm nach einem ON-ER-ROR-GOTO-Befehl mit der Programmzeile »Zeile« fort
RESUME NEXT	Setze Programm nach einem ON-ER-ROR-GOTO-Befehl mit der Zeile fort, die der fehlerverursachenden Zeile folgt
RETURN [<Zeile>]	Kehre aus dem Unterprogramm in das übergeordnete Programm [zur Zeile »Zeile«] zurück
RMDIR <Pfad>	Lösche das (leere) Verzeichnis »Pfad« von Diskette oder Festplatte
RSET <Variable> = <String>	Lege die Daten der Zeichenkette »String« rechtsbündig in dem durch den Field-Befehl definierten Datenfeld der String-Variablen »Variable« ab
RUN [<Zeile>]	Starte das Basic-Programm [ab Programmzeile »Zeile«]
RUN <Datei> [,R]	Lade und starte das Programm »Datei« [und lösche zuvor alle noch im Speicher befindlichen Programmzeilen, schließe jedoch nicht eventuell geöffnete Dateien]
SAVE <Datei> [,A] [,P]	Schreibe ein Basic-Programm unter dem Namen »Datei« auf Diskette oder Festplatte [im ASCII-Format] [in geschützter Form]
SCREEN [<Modus>] [, <Darstellung>] [, <Seite1>] [, <Seite2>]	Wähle [den Bildschirmmodus »Modus«], [in farbiger oder Schwarzweiß-Darstellung], [Bildschirmseite »Seite1« als für Text- und Grafikausgaben zuständige Seite] [und Bildschirmseite »Seite2« als sichtbare Seite]
SHELL [<String>]	Rufe die MS-DOS-Befehlsebene auf und erwarte die Eingabe von MS-DOS-Befehlen (Rückkehr in den GW-Basic-Interpreter über den Befehl »exit«) [führe den Befehl »String« aus und kehre sofort in Basic zurück]

Befehl und Parameter	Bedeutung
SOUND <Frequenz> , <Dauer>	Erzeuge einen Ton der Frequenz »Frequenz« mit einer Dauer von »Dauer« Systemtakt
STOP	Unterbreche Programm (Fortsetzung mit dem Befehl CONT)
STRIG(<Knopf>) [ON] [OFF] [STOP]	Erlaube ([ON]), sperre ([OFF]) oder speichere ([STOP]) Unterbrechungsanforderungen der Joy-sticktaste »Knopf« (0, 2, 4 oder 6)
SWAP <Variable1> , <Variable2>	Tausche die Werte der Variablen »Variable1« und »Variable2« (gleichen Typs) gegeneinander aus
SYSTEM	Schließe alle Dateien, verlasse den GW-Basic-Interpreter und kehre in die MS-DOS-Befehlsebene zurück
TIMES\$ = " <Zeit> "	Setze die Systemuhr auf die Zeit »Zeit«
TIMER [ON] [OFF] [STOP]	Erlaube ([ON]), sperre ([OFF]) oder speichere ([STOP]) Unterbrechungsanforderungen der Systemuhr
TROFF	Schalte Trace-Funktion aus
TRON	Schalte Trace-Funktion ein (gebe während eines Programmlaufs die Zeilennummer jeder bearbeiteten Programmzeile auf den Bildschirm aus)
USR [<n>] (<Variable>)	Rufe ein Maschinencodeprogramm mit der durch »def usr [n]« definierten Startadresse auf und übergebe den Wert der Variablen »Variable«
VIEW [SCREEN] [<x1> , <y1> - <x2> , <y2>] [, <Farbe>] [, <Rahmen>]]]	Definiere ein Grafikfenster [mit den angegebenen Eckkoordinaten [in der Farbe »Farbe« [und zeichne einen Rahmen in der Farbe »Rahmen«]]. Ist »SCREEN« angegeben, erhält die linke obere Ecke des neuen Fensters die Grafikkoordinaten 0,0
VIEW PRINT [<Zeile1> TO <Zeile2>]	Definiere ein Textfenster [von Textzeile »Zeile1« bis Textzeile »Zeile2«]
WAIT <Offset> , <Wert1> [, <Wert2>]	Warte, bis der über die Portadresse »Offset« anliegende Bytewert [verknüpft mit »Wert2« über die logische XOR-Funktion], verknüpft mit »Wert1« über die logische AND-Funktion ein Ergebnis ungleich Null ergibt. Läßt sich nicht durch [Ctrl-Break] abbrechen
WEND	Ende einer WHILE-WEND-Schleife
WHILE <Argument>	Führe die bis zum nächsten »WEND« folgenden Befehle als Programmschleife aus, solange der Vergleich von »Argument« wahr (ungleich Null) ist
WIDTH [40] [80]	Begrenze die Anzahl der Zeichen pro Zeile bei der Bildschirmausgabe auf 40 oder 80 Zeichen
WIDTH # <Kanal> , <Breite>	Begrenze die Anzahl der Zeichen pro Zeile bei der Dateiausgabe über Kanal »Kanal« auf »Breite« Zeichen
WIDTH <Gerät> , <Breite>	Begrenze die Anzahl der Zeichen pro Zeile bei der Ausgabe über Gerät »Gerät« (zum Beispiel »prn«) auf »Breite« Zeichen
WIDTH LPRINT <shortint>	Begrenze die Anzahl der Zeichen pro Zeile bei der Druckerausgabe auf »shortint« Zeichen
WINDOW (<x1> , <y1>)- (<x2> , <y2>)	Neudefinierung der Eckpunkte links oben (x1,y1) und rechts unten (x2,y2) des Grafikbildschirms für eine Skalierung (Neueinteilung der Einheiten)
WINDOW SCREEN (<x1> , <y1>)-(<x2> , <y2>)	Neudefinierung der Eckpunkte links unten (x1,y1) und rechts oben (x2,y2) des Grafikbildschirms für eine Skalierung (Neueinteilung der Einheiten)
WRITE <Zeichen>	Gebe Text und/oder Variablen »Zeichen« auf dem Bildschirm aus, trenne Werte durch Kommata und schließe Zeichenketten in Anführungszeichen ein
WRITE # <Kanal> , <Zeichen>	Gebe Text und/oder Variablen »Zeichen« über den Dateikanal »Kanal« aus, trenne Werte durch Kommata und schließe Zeichenketten in Anführungszeichen ein
<Argument1> XOR <Argument2>	Verknüpfe »Argument1« mit »Argument2« über die logische Exklusiv-Oder-Funktion

Tabelle 1. Die Basic-Steuerbefehle

Funktion und Parameter	Bedeutung
ABS (<Zahl>)	Absolutwert von »Zahl« (ignoriere Vorzeichen)
ASC (<String>)	ASCII-Wert des ersten Zeichens von »String«. Darf nicht leer (0) sein
ATN (<Winkel>)	Arkustangens des im Bogenmaß angegebenen Wertes »Winkel«
CDBL (<Wert>)	Einfachgenauer Wert von »Wert« in einen doppeltgenauen Wert umgewandelt
CHR\$ (<Zeichen>)	ASCII-Zeichen von »Zeichen«
CINT (<Wert>)	Gerundeter Integer-Wert von »Wert«
COS (<Winkel>)	Kosinus des Winkels »Winkel« (im Bogenmaß)
CSNG (<Wert>)	Doppeltgenauer Wert von »Wert« in einen einfachgenauen Wert umgewandelt
CVD (<String>)	Numerischer Wert der 8-Byte-Zeichenkette »String«
CVI (<String>)	Numerischer Wert der 2-Byte-Zeichenkette »String«
CVS (<String>)	Numerischer Wert der 4-Byte-Zeichenkette »String«
ENVIRON\$ (<Nummer>)	Parameter Nummer »Nummer« der Betriebssystemumgebung
ENVIRON\$ (<String>)	Parameter »String« der Betriebssystemumgebung
EOF (<Kanal>)	Zeigt durch den Wert -1 (logisch wahr) das Ende der Datei in Kanal »Kanal« an (ansonsten auf 0 (logisch unwahr) gesetzt)
EXP (<Wert>)	Basis der natürlichen Logarithmen (e) potenziert mit »Wert«
FIX (<Wert>)	Vorkommateil von »Wert« (keine Rundung)
FN <Funktion> [(<Variablen>)]	Ergebnis der durch DEF FN »Funktion« definierten Funktion
HEX\$ (<Dezimal>)	Hexadezimaler Wert von »Dezimal«
INP (<Offset>)	An Portadresse »Offset« anliegender Bytewert
INPUT\$ (<Länge> [, # <Kanal>])	Nächste Zeichenkette von der Tastatur [vom Dateikanal »Kanal«] der Länge »Länge« Bytes
INSTR ([<Position> ,] <String1> , <String2>)	Position, an der »String2« in »String1« enthalten ist [ab Position »Position« in »String1«]
INT (<Wert>)	Ganzzahliger Anteil von »Wert«
IOCTL\$ (# <Kanal>)	Über den Dateikanal »Kanal« von einem Gerätetreiber eingelesene Steuerzeichenkette (ab Basic-Version 3.0)
LEFT\$ (<String> , <Position>)	Die ersten »Position« Zeichen der Zeichenkette »String«
LEN (<String>)	Anzahl der Zeichen im Text »String«
LOC (<Kanal>)	Anzahl der gelesenen oder geschriebenen Datenblöcke (zu je 128 Bytes) von/auf Dateikanal »Kanal«
LOF (<Kanal>)	Länge der Datei, die über Kanal »Kanal« angesprochen ist, in Byte. Ist »Kanal« auf einer seriellen Schnittstelle geöffnet, gibt »lof« die Anzahl der noch freien Bytes im RS232-Puffer zurück
LOG (<Zahl>)	Natürlicher Logarithmus von »Zahl«
MID\$ (<String> , <Von> [, <Länge>])	Zeichenkette aus »String« ab der Position »Von« bis zum Textende [oder mit der Länge »Länge«]
MKD\$ (<Wert>)	Der in eine Zeichenkette geänderte doppeltgenaue Wert von »Wert«
MKI\$ (<Wert>)	Der in eine Zeichenkette geänderte ganzzahlige Wert von »Wert«
MKS\$ (<Wert>)	Der in eine Zeichenkette gewänderte einfache-genaue Wert von »Wert«
OCT\$ (<Dezimal>)	Oktaler Wert von »Dezimal«

Tabelle 2. Die Basic-Funktionen

Funktion und Parameter	Bedeutung
PEEK (<Offset>)	Inhalt der Speicherzelle »Offset«
PEN (<Funktion>)	Ergebnis der Lichtgriffel-Funktion »Funktion«
PMAP (<Funktion>)	Koordinatenumrechnung für ein mit »WINDOW« definiertes Fenster (siehe entsprechenden Artikel in diesem Sonderheft)
POINT (<x>, <y>)	Farbe der Bildschirmkoordinate (x,y)
POINT (<Modus>)	x- oder y-Anteil der Bildschirm- oder Anwenderkoordinate (abhängig von »Modus«)
RIGHT\$ (<String>, <Länge>)	Die letzten »Länge« Zeichen der Zeichenkette »String«
RND (<Wert>)	Zufallszahl zwischen 0 und 1 (»Wert« = 0: Wiederholung der letzten Zufallszahl)
SCREEN (<Zeile>, <Spalte> [, <Funktion>])	ASCII-Code des Zeichens auf der Textkoordinate (xtext,ytext) [»Funktion«=1: Übergebe statt dessen Farbattribut]
SGN (<Wert>)	Wenn »Wert« > 0: 1, wenn »Wert« < 0: -1 und wenn »Wert« = 0: 0
SIN (<Winkel>)	Sinus des Winkels »Winkel« (im Bogenmaß)
SPACE\$ (<Anzahl>)	Zeichenkette, die aus »Anzahl« Leerzeichen besteht
SQR (<Wert>)	Quadratwurzel von »Wert«
STICK (<Joystick>)	Numerischer Wert der Joystick-Koordinate »Joystick« (0 = X-Wert von Joystick A, 1 = Y-Wert, 2 = X-Wert von Joystick B, 3 = Y-Wert von B)
STR\$ (<Zahl>)	Wert von »Zahl« in eine Zeichenkette umgewandelt
STRIG (<Status>)	Status der Joysticktasten
STRING\$ (<Länge>, <Wert>)	Zeichenkette, die aus »Länge« Zeichen mit dem ASCII-Code »Wert« oder dem ersten Zeichen des Strings »Wert« zusammengesetzt ist
TAN (<Winkel>)	Tangens des Winkels »Winkel« (im Bogenmaß)

Funktion und Parameter	Bedeutung
VAL (<String>)	Numerischer Wert der Zeichenkette »String«
VARPTR (<Variable>)	Startadresse (Offset) der Variablen »Variable« im Basic-Datensegment
VARPTR (# <Kanal>)	Startadresse des Dateisteuerblocks für Dateikanal »Kanal«
VARPTR\$ (<Variable>)	3-Byte-Zeichenkette, deren erstes Zeichen den Variablentyp (2: Integer, 3: String, 4: Real, 8: Doppelt genau), deren zweites Zeichen den niederwertigen und deren drittes Zeichen den höherwertigen Teil der Startadresse im Arbeitsspeicher angibt

Tabelle 2. Die Basic-Funktionen

Systemvariable	Bedeutung
CSRLIN	Vertikale Position, auf der sich der Text-Cursor befindet
DAT\$	Datum der Systemuhr
ERDEV	Fehlercode des peripheren Gerätes, das den letzten Fehler gemeldet hat
ERDEV\$	Name des Gerätetreibers, dessen Gerät den letzten Fehler gemeldet hat
ERL	Nummer der Programmzeile, in der der letzte Fehler aufgetreten ist
ERR	Fehlercode des Fehlers, der zuletzt aufgetreten ist
FRE(0)	Größe des von Basic nicht benutzten Speicherbereichs
FRE(““)	Größe des von Basic nicht benutzten Speicherbereichs mit Auslösung einer Garbage Collection
INKEY\$	Zeichen der zuletzt gedrückten Taste
LPOS (0/1/2/3)	Horizontale Position des Druckkopfes innerhalb des Druckpuffers von Drucker 1 (0/1), Drucker 2 (2) oder Drucker 3 (3)
PLAY (0)	Anzahl der Noten in der Tonwarteschlange
POS (0)	Horizontale Position, auf der sich der Text-Cursor befindet
TIM\$	Zeit der Systemuhr
TIMER	Anzahl der Sekunden, die seit Mitternacht verstrichen sind

Tabelle 3. Die Basic-Systemvariablen

Wissen Sie mehr von PCs als andere Leute?

Dann schreiben Sie uns. Die Redaktion DOS International sucht aktive Mitarbeiter, die über interessante Gebiete aus der »Computerei« berichten wollen. MS-DOS, DOS Plus, GEM, Drucker, Basteleien – es gibt nichts, worüber nicht andere Leser unserer Zeitschrift auch etwas lesen wollen.

Ihren Einsatz lassen wir uns natürlich etwas kosten. Ihr Manuskript wird selbstverständlich honoriert. Bilder, Zeichnungen, Tabellen und Listings werden gesondert berechnet.

Doch bevor Sie sich hinsetzen und das Schreiben anfangen: Rufen Sie uns an. Sagen Sie uns, was Sie schreiben wollen, und wir klären ob – und wann – der Artikel in unser Konzept paßt.

Der Artikel muß mit einem Textsystem erfaßt werden. Wir brauchen von Ihnen eine Diskette im MS-DOS-Format, auf der der Text als reines ASCII-Feld steht.

Schreiben Sie uns auch, wenn Sie nur Vorschläge für Themen haben. DOS International ist nämlich eine Zeitschrift von aktiven PC-Besitzern für aktive PC-Benutzer. Besonders das DOS-Forum lebt von Ihrer Mitarbeit. Auch die Redaktion kann nicht alle Antworten wissen. Fragen an uns, die nicht sofort beantwortet werden können, werden im DOS-Forum veröffentlicht. Wissen Sie eine Antwort, so wird diese veröffentlicht.

Sie sehen: DOS International ist die PC-Zeitschrift, in der das Wissen der PC-Benutzer untereinander ausgetauscht wird.

Redaktion · DOS International · Hans-Stießberger-Str. 3 · 8013 Haar/München · Tel.: (089) 46 10 01-0

ABO N M E N T



Ein Abonnement ist praktisch und gewährt zusätzlich noch Preisvorteile

Das DOS-Abonnement ist preiswert zu haben:

Im europäischen Ausland:

6 Ausg. = 50,- DM

12 Ausg. = 100,- DM

Im Inland bzw. West-Berlin

6 Ausg. = 40,- DM

12 Ausg. = 80,- DM

Im außereuropäischen Ausland:

6 Ausg. = 75,- DM

12 Ausg. = 150,- DM

**Bitte benutzen Sie
die Bestellkarte**

Garantie: Wir garantieren jedem Abonnenten das Recht, seine Bestellung innerhalb einer Woche nach Abschluß schriftlich zu widerrufen. Die rechtzeitige Absendung des Widerrufschreibens genügt zur Fristwahrung.

DMV-Verlag · Postfach 250 · 3440 Eschwege

GW-Basic-Referenz: Programmablauf-Steuerung

Die wichtigsten Basic-Befehle dienen der Steuerung des Programmablaufs. Diese Kommandos ändern direkt einen Zeiger auf eine bestimmte Zeilennummer.

CONT

Führt die Programmausführung nach einem Stop- oder End-Befehl oder nach dem Drücken der Tastenkombination [Ctrl-Break] bei der nächsten Anweisung weiter. Beachten Sie, daß nach Änderungen im Programmtext ein Fortfahren nicht mehr möglich ist.

STOP

Beendet die Programmausführung sofort. Im Gegensatz zum End-Befehl schließt der Interpreter jedoch keine zu diesem Zeitpunkt noch geöffneten Dateien. Nach dem Stop-Befehl befindet sich der Interpreter im Direktmodus, auch wenn das Programm als Parameter zu »gwbasic.exe« sofort gestartet wurde. Alle Variableninhalte bleiben erhalten und lassen sich mit dem Print-Befehl abfragen. Solange noch keine Änderungen im Programmtext erfolgt sind, startet der Cont-Befehl das Programm bei der Anweisung, die dem Stop-Kommando folgt. Wiederum bleiben die Variablenwerte erhalten.

»Stop« läßt sich sehr gut zur Fehler-suche (»Debugging«) bei der Programm-entwicklung einsetzen. Wenn Sie beispielsweise den Verdacht haben, daß eine bestimmte Zeile nie erreicht wird, setzen Sie an den Anfang dieser Zeile eine Stop-Anweisung. Die Meldung des Interpreters lautet dann »break in <Zeile>«.

END

Beendet sofort die Programmausführung, die Sie mit »cont« eventuell fortsetzen können. Im Gegensatz zum Stop-Befehl gibt GW-Basic jedoch die Zeilennummer, in der die End-Anweisung steht, nicht aus. Außerdem werden sämtliche offenen Datenkanäle geschlossen.

Ein End-Befehl in der letzten Programmzeile, wie in manchen Lehrbüchern angegeben, ist bei GW-Basic nicht erforderlich.

FOR <Schleifenzähler> =
<Startwert> TO <Endwert>
[STEP <Schrittweite>]
NEXT [<Schleifenzähler>]
[,<Schleifenzähler>] [...]

Führt eine Programmschleife zwischen der For- und der Next-Zeile solange aus, bis der Inhalt der Varia-

blen »Schleifenzähler« entweder größer oder gleich (bei positiver Schrittweite) oder kleiner oder gleich dem Endwert ist (bei negativer Schrittweite). Der Standardwert für die Schrittweite beträgt (+)1.

Zu jeder For-Anweisung muß genau eine Next-Zeile gehören. Der besseren Übersicht wegen sollte man bei mehreren ineinander geschachtelten Schleifen den Schleifenzähler beim Next-Kommando mit angeben. GW-Basic selbst verlangt dies nicht, weil bei zum Beispiel drei geschachtelten Schleifen die nächste Next-Anweisung automatisch auf die zugehörige For-Zeile weist. Schleifen dürfen nur über die Next-Anweisung und nicht über einen Goto- oder Gosub-Aufruf verlassen werden. Möchten Sie eine Schleife dennoch vorzeitig beenden, dann setzen Sie den Schleifenzähler auf den Endwert und führen einen Goto-Befehl auf die zugehörige Next-Zeile aus.

Der genaue Ablauf einer For-Next-Anweisung gliedert sich folgendermaßen: Zuerst setzt der Interpreter den Inhalt des Schleifenzählers auf den Startwert. Ist die Schrittweite negativ und der Schleifenzähler kleiner als der Endwert, dann wird der Programmteil bis zur zugehörigen Next-Anweisung übersprungen und die Schleife beendet. Dies trifft ebenso zu, wenn die Schrittweite positiv und der Schleifenzähler größer als der Endwert ist. In allen anderen Fällen erhöht, beziehungsweise erniedrigt der Interpreter den Schleifenzähler um die Schrittweite und führt die Kommandos bis zum nächsten Next-Befehl aus.

Keinesfalls sollten For-Next-Schleifen wie folgt geschachtelt werden:

```
10 for x=1 to 10
20 for y=1 to 10
30 next x
40 next y
```

Dies hätte eine sofortige Fehlermeldung (»next ohne for in 40«) zur Folge, da der Interpreter die y-Schleife nicht korrekt bearbeiten kann.

Achten Sie auch darauf, in einer laufenden Schleife keine erneute For-Anweisung mit dem gleichen Schleifenzähler zu setzen.

GOTO <Zielzeile>

GW-Basic führt als nächste Anweisung den ersten Befehl der angegebenen Zielzeile aus. Alle Befehle hinter dem Goto-Kommando werden ignoriert. Beispiel:

```
10 goto 100 : print "test"
```

Die Anweisung »print« würde hier niemals ausgeführt, weil das Programm zuvor zu Zeile 100 verzweigt. Die Angabe einer Variablen anstatt einer Zahl für die Zielzeile ist ebenso wie beim Gosub-Befehl nicht erlaubt.

GOSUB <Zielzeile>

RETURN

Setzt die Programmausführung ab der Zielzeile fort, bis ein Return-Befehl folgt. Zwischen dieser Zeile und der Return-Anweisung steht ein Unterprogramm, das meist von mehreren Stellen des Hauptprogramms benötigt wird. Nach Erreichen von »return« setzt der Interpreter die Programmausführung an der dem Gosub-Befehl folgenden Anweisung fort.

Üblicherweise setzt man Unter-routinen ans Programmende. Achten Sie dann jedoch darauf, daß Ihr Hauptprogramm mit einer End- oder Stop-Anweisung beendet wird, sonst »läuft« der Interpreter zum Schluß noch einmal in die Unter-routine. Ein Beispiel dafür, wie es nicht gemacht werden darf:

```
10 rem Unterprogramm-Test
20 print "Hier ist das Hauptprogramm"
30 gosub 100
40 print "Jetzt ist das Hauptprogramm zu Ende"
100 print "Hier ist das Unterprogramm"
110 return
```

Dies ergibt die Fehlermeldung »return ohne gosub in 110« am Ende des Programmlaufs, da die Unter-routine nach Ausführung von Zeile 40 noch einmal abgearbeitet wird. Es fehlt eine End-Anweisung zwischen den Zeilen 40 und 100 (zum Beispiel »50 end«).

Ein weiterer, häufiger Fehler liegt darin, daß ein Unterprogramm für eigene Berechnungen dieselben Variablenamen verwendet, die im Hauptprogramm für andere Zwecke benötigt werden. Solche Fehler sind vor allem in längeren Programmen mit vielen Unter-routinen nur schwer zu entdecken. Man beugt ihnen am besten durch eine handschriftliche Liste aller verwendeten Variablen und deren Bedeutung vor.

Sowohl »goto« als auch »gosub« können Sie im Direktmodus eingeben, um ein Programm an einer bestimmten Stelle fortzusetzen.

IF <Bedingung> THEN <Anweisung1> [ELSE <Anweisung2>]

Innerhalb der Bedingung steht ein Vergleich, dessen Auswertung den weiteren Verlauf der Zeile bestimmt. Die Bedingung ist üblicherweise eine Datenüberprüfung einer Variablen. Beispiel:

```
10 if a=10 then print "a ist 10!" else
print "a ist ungleich 10!"
```

Diese Zeile ist durch die Print-Texte selbsterklärend. Als Vergleichsoperatoren sind zulässig: »>« (größer als), »<« (kleiner als), »> =« (größer oder gleich), »< =« (kleiner oder gleich) und »< >« (verschieden von).

Eine Bedingung darf auch aus mehreren Abfragen bestehen, die durch die Operatoren »and« oder »or« verknüpft sind. Beispiele:

```
10 if a=10 and b=20 then print "a ist 10
und b ist 20!"
```

```
20 if a=10 or b=20 then print "entweder
ist a 10 oder b ist 20!"
```

Das Ergebnis einer Bedingung können Sie durch Voranstellen des Schlüsselwortes »not« negieren. Auch hierzu wieder ein Beispiel:

```
10 if not a=10 then print "a ist nicht 10!"
```

Allerdings wäre diese Zeile gleichbedeutend mit

```
10 if a < > 10 then print "a ist nicht 10!"
```

Letztere Darstellung ist jedoch der Not-Methode vorzuziehen, weil sie leichter lesbar ist.

Die Anweisung hinter dem »else« (sofern angegeben) führt GW-Basic immer dann aus, wenn eine Überprüfung der Bedingung logisch unwahr ist. Sowohl »Anweisung1« als auch »Anweisung2« dürfen aus mehreren, durch Doppelpunkte getrennte Befehlen bestehen. Beispiel:

```
10 if a=0 then a=1 : b=1 : c=1 : else print
"a ungleich 0!" : d=1
```

Wichtig ist hier vor allem der letzte Doppelpunkt vor dem Else-Befehl. Wenn »a« den Wert 0 hat, setzt GW-Basic die Variablen »a«, »b« und »c« auf 1, ansonsten gibt es den Text »a ungleich 0!« aus und setzt »d« auf 1. Wenn eine der Anweisungen ein Goto-Befehl wäre, darf er auch fehlen.

```
10 if a=10 then goto 100
und
```

```
10 if a=10 then 100
```

sind damit gleichbedeutend.

ON <Variable> GOTO | GOSUB <Zeile1> [, <Zeile2>] [...]

Hier verzweigt der Interpreter in Abhängigkeit des Inhalts der Variablen in eine der aufgelisteten Zeilennummern. Steht die Variable auf 1, so wird zur Zeile1 verzweigt, bei 2 zur Zeile2 und so weiter. Dabei kann es sich wahlweise um einen Goto- oder einen GOSUB-Aufruf handeln. Sollte

die Variable leer sein, also den Inhalt 0 haben, oder größer als 255 sein, dann ignoriert GW-Basic den gesamten Rest der Zeile (auch eventuell nachfolgende Kommandos) und fährt mit der Programmausführung in der nächsten Zeile fort. Gegebenenfalls wird der Inhalt der Variablen auf einen ganzzahligen Wert gerundet.

Mit dieser Methode lassen sich trickreiche Wert-Abfragen programmieren. Angenommen, Sie möchten in Abhängigkeit von der Variablen »a« ein Kommando ausführen. Dies soll nur dann der Fall sein, wenn »a« den Inhalt 2, 3 oder 5 angenommen hat:

```
50 on a goto 80,70,80,80,70
60 goto 70
70 print "a ist jetzt 2, 3 oder 5"
80 ...
```

Bei den Werten 1 und 4 sowie allen anderen Zahlen fährt GW-Basic mit Zeile 80 fort, ansonsten mit Zeile 70.

READ <Variable> [, <Variable>] DATA <Element1> [, <Element2>] ,...

Eine Read-Anweisung holt sich immer das nächste Element einer mit dem Schlüsselwort »data« beginnenden Programmzeile und speichert es in der Variablen. Dies können sowohl Zeichenketten als auch Zahlen sein. Soll in einer Zeichenkette einer Data-Zeile ein Komma Verwendung finden, dann muß das jeweilige Element innerhalb von Anführungszeichen stehen. Beispiel:

```
10 read a$,b$ : read c
20 print a$ : print b$ : print c
100 data "mit Komma: ",hallo,1234
```

In diesem Fall weist GW-Basic der Variablen »a\$« den Text »mit Komma:«, »b\$« den Text »hallo« zu. Die Zahlenvariable »c« erhält den Inhalt 1234. Beachten Sie, daß eine Data-Anweisung immer als letzte in einer Zeile stehen muß. Nachfolgende Befehle sieht der Interpreter als weitere Data-Elemente an.

Eine Read-Anweisung kann so lange ausgeführt werden, bis keine Elemente mehr zum Lesen zur Verfügung stehen. Ein interner Zeiger von GW-Basic deutet jeweils auf das nächste Element.

RESTORE [<Zeile>]

Setzt den Data-Zeiger auf die erste Programmzeile, die eine Data-Anweisung enthält, beziehungsweise auf die angegebene Zeile. Die nächste Read-Anweisung liest dann das erste Element der neuen Zeile.

REM (oder »')«

Der gesamte Rest der Zeile ab der Rem-Anweisung wird vom Interpreter übergangen. Damit lassen sich Bemerkungen in den Programmtext ein-

fügen. Wenn Sie den Apostroph zur Einleitung einer Bemerkung verwenden, darf der Doppelpunkt vor diesem Zeichen fehlen. Beispiele:

```
10 a=10 : rem A erhält den Inhalt 10
20 b=20 'B erhält den Inhalt 20
```

»Rem« und »'« müssen immer als letzte beziehungsweise einzige Befehle in einer Zeile stehen.

RUN [<Zeile>] | [<Dateiname> [,<R>]]

Führt den Befehl »clear« aus und startet ein Programm ab der angegebenen Zeile oder lädt ein Basic-Programm mit dem jeweiligen Dateinamen und startet es sofort ab der ersten Zeile. Der Parameter »R« bewirkt hierbei, daß eventuell geöffnete Dateien vom Clear-Befehl nicht geschlossen werden und so dem neuen Programm weiter zur Verfügung stehen.

SHELL ["<MS-DOS-Befehl>"]

»Shell« erlaubt die direkte Ausführung von beliebigen MS-DOS-Kommandos. Allerdings dürfen dabei keine speicherresidenten Programme initialisiert werden, weil der von GW-Basic belegte Speicher sonst nicht mehr zur Verfügung steht. Es ist ebenfalls nicht erlaubt, GW-Basic von der nachgeladenen Kommandoebene aus ein zweites Mal zu starten. »Shell« ohne Parameter lädt den Kommandointerpreter »command.com«, wobei Sie dann durch den Befehl »exit« wieder ins Basic zurückgelangen.

SYSTEM

»System« beendet die Sitzung mit GW-Basic. Wenn Sie »system« als Befehl innerhalb eines Programms verwenden möchten, bedenken Sie beim Testen unbedingt, daß alle Programmänderungen hiermit verloren sind. Am besten setzen Sie einen entsprechenden Save-Befehl vor das System-Kommando.

TRON | TROFF

Dienen dem Ein- (»tron«) und Ausschalten (»troff«) des »Trace-Modus« von GW-Basic. Innerhalb dieses Modus wird jede gerade ausgeführte Zeilennummer auf dem Bildschirm dargestellt, wodurch sich auch die Programmausführung deutlich verlangsamt. Allerdings haben Sie keinerlei Möglichkeit, den Programmablauf selbst zu beeinflussen, also zum Beispiel kurz anzuhalten und Variablenwerte einzusehen. Obendrein schreibt der Interpreter die Zeilennummern fortlaufend hintereinander, so daß Beobachtungen von korrekten Locate- und Print-Anweisungen leider unmöglich sind.

(tr)

GW-Basic-Referenz: Grafik

GW-Basic unterstützt normalerweise nur Farbgrafikkarten und den Monochromadapter. Hercules-Grafiken können nur speziell modifizierte Interpreter bearbeiten. Eigentlich schade, denn der Grafikbefehlssatz ist vielfältig.

Um überhaupt Grafiken unter GW-Basic erzeugen zu können, benötigen Sie eine CGA- oder EGA-Grafikkarte. Hercules-Karten werden nicht unterstützt, jedoch gibt es von einigen Herstellern leicht modifizierte Basic-Versionen, die auch mit dieser Grafikauflösung arbeiten.

Mit einer CGA-Karte haben Sie zwei verschiedene Grafikmodi zur Verfügung:

CGA-Modus 1:

320x200 Punkte, vier Farben

CGA-Modus 2:

640x200 Punkte, zwei Farben

Eine EGA-Karte bietet die genannten zwei und drei weitere Modi:

EGA-Modus 1:

320x200 Punkte, 16 Farben

EGA-Modus 2:

640x200 Punkte, 16 Farben

EGA-Modus 3:

640x350 Punkte, 16 Farben

Welche Modi Sie auf Ihrem PC einsetzen können, hängt natürlich auch vom verwendeten Monitor ab. Eine zum Beispiel an einem Monochrom-Monitor betriebene EGA-Karte stellt keinerlei Farben dar.

Zum Einschalten der Grafik dient der Basic-Befehl »screen«. Die genaue Schreibweise lautet:

**SCREEN [<Modus>] [<Farbe>]
[,<Seite1>] [<Seite2>]**

Für den Modus müssen Sie zwischen einer CGA- und einer EGA-Karte unterscheiden. Eine CGA-Karte kennt die folgenden Modi:

0 Textmodus

1 Grafik

mit 320x200 Punkten in 4 Farben

2 Grafik

mit 640x200 Punkten in 2 Farben

Bei einer EGA-Karte sind zusätzlich folgende Modi erlaubt:

7 Grafik

mit 320x200 Punkten in 16 Farben

8 Grafik

mit 640x200 Punkten in 16 Farben

9 Grafik

mit 640x350 Punkten in 16 Farben

10 Grafik

mit 640x350 Punkten in 2 Farben

(monochrom)

Die Farbangabe darf die Werte 0 und 1 annehmen. Bei 1 übernimmt GW-Basic beim Einschalten der Grafik die aktuellen Farbwerte und bei 0 setzt es die Standardfarben Schwarz und Weiß.

Die beiden Parameter »Seite1« und »Seite2« sind nur bei einer EGA- oder

VGA-Karte von Bedeutung, weil diese mehrere Bildschirmseiten gleichzeitig verwalten können. »Seite1« gibt die Seitennummer (ab 0 gezählt) an, die auf dem Bildschirm angezeigt werden soll und »Seite2« spezifiziert die Seite, auf die sich alle Ausgabebefehle wie zum Beispiel »print« beziehen sollen. Die Anzahl der maximal zulässigen Seiten hängt von der Speicherausbaustufe Ihrer EGA- oder VGA-Grafikkarte ab:

64 KByte: 2 Seiten im 320x200-

Punkte-Modus, 1 Seite im 640x200-

Punkte-Modus

128 KByte: 4 Seiten im

320x200-Punkte-Modus, 2 Seiten im

640x200-Punkte-Modus

256 KByte: 8 Seiten im

320x200-Punkte-Modus, 4 Seiten im

640x200-Punkte-Modus, 2 Seiten im

640x350-Punkte-Modus

Durch Umschalten können Sie beispielsweise auf einer nicht sichtbaren Seite eine Grafik aufbauen. Ist die entsprechende Routine fertig, dann wechseln Sie die Darstellungsart.

Zu einer anspruchsvollen Grafik gehören natürlich auch verschiedene Farben

Die Definition der Farben für den gewählten Grafikmodus erfolgt mit dem Color-Befehl. Die Schreibweise lautet:

**COLOR [<Vordergrundfarbe>]
[,<Hintergrundfarbe>]**

Bei einer CGA-Karte im hochauflösenden Modus (»screen 2«) sind normalerweise nur Schwarz (0) und Weiß (1) als Werte zulässig. Mit einem Trick sind aber auch bis zu 16 Farben darstellbar, einen entsprechenden Beitrag finden Sie in der DOS Extra 5, Seite 23, »16 Farben für die CGA-Karte«. Bei einer CGA-Karte im niedrigauflösenden Modus sind Werte von 0 bis 3, beziehungsweise bei einer EGA- oder VGA-Karte Werte von 0 bis 15 erlaubt.

Der niedrigauflösende Modus (»screen 1«) der CGA-Karte bietet zwei Paletten zu je vier Farben zur Auswahl. Dabei ändert sich die Syntax des Color-Befehls:

**COLOR [<Vordergrundfarbe>]
[,<Palette>]**

CGA-Palette 0:

0 Hintergrundfarbe (gesetzter Punkt nicht sichtbar)

1 Grün

2 Rot

3 Gelb

CGA-Palette 1:

0 Hintergrundfarbe (gesetzter Punkt nicht sichtbar)

1 Cyan

2 Magenta

3 Weiß

Bei einer EGA- oder VGA-Grafikkarte sind insgesamt 16 Farben darstellbar. Hier bietet eine Palette nicht 4, sondern 16 Farben, die zusätzlich noch frei wählbar sind. Für jeden der 16 Farbspeicher der Palette können Sie eine der 64 möglichen Farben festlegen. Die Syntax des Palette-Befehls lautet:

**PALETTE <Farbnummer> ,
<Farbe>**

Für die Farbnummer sind Werte von 0 bis 15 und für die Farbe Werte von 0 bis 63 erlaubt. Dies gilt, wie bereits erwähnt, für eine EGA- und VGA-Grafikkarte.

Wählen Sie aus vier Farben bei einer CGA- und aus 16 Farben bei einer EGA-Karte

Möchten Sie alle 16 Farbspeicherplätze mit neuen Farben belegen, bietet sich der platzsparende Palette-Using-Befehl an:

PALETTE USING <Feldname> (<Startindex>)

Dazu definieren Sie vor dem Befehl eine Feldvariable (Array) mit 16 Plätzen und belegen diese mit den gewünschten Farbnummern (0 bis 63). Der Startindex teilt GW-Basic mit, ab welcher Stelle des Variablenfeldes »Feldname« es die entsprechenden Werte suchen soll, also zum Beispiel »palette using farben(1)«, wenn das Variablenfeld »farben(...)« von Position 1 bis 16 die neu zu setzenden Farbwerte enthält.

Nachdem der Grafikmodus gesetzt ist und die Farben gewählt wurden, kann das Programm den ersten Grafikpunkt setzen. Der einfachste Grafikbefehl lautet »pset« und setzt einen einzelnen Punkt auf dem Bildschirm.

PSET [STEP] (<x>,<y>) [<Zeichenfarbe>]

Ist der Parameter »step« angegeben, dann sieht GW-Basic die Koordinaten nicht als absolut, sondern als relativ zum zuletzt gezeichneten Punkt, beziehungsweise zum Bildschirmmittelpunkt, falls dies der erste Zeichenbefehl ist. Dies gilt auch für alle nachfolgenden Grafikbefehle, die diese Option besitzen.

»x« und »y« sind die horizontalen und vertikalen Koordinaten, an denen der Interpreter den angeforderten

Punkt setzt. Je nach gewählter Auflösung sind für »x« Werte zwischen 0 und 199, 399 oder 349 zulässig. »y« darf zwischen 0 und 319 oder 639 variieren.

Die Zeichenfarbe gibt die Farbnummer aus der Farbpalette an. Fehlt diese Angabe, dann setzt GW-Basic den Punkt in der mit dem Color-Befehl festgelegten Vordergrundfarbe. Zum Löschen eines Bildpunktes dient der Preset-Befehl. Seine Syntax ist mit der des Pset-Kommandos identisch und lautet:

PRESET [STEP] (<x>, <y>)
[, <Zeichenfarbe>]

Der Unterschied zu »pset« besteht darin, daß bei fehlender Angabe der Zeichenfarbe der Punkt in der Hintergrundfarbe gesetzt, also gelöscht wird.

Der Befehl zum Zeichnen von Linien und Rechtecken lautet »line«. Seine Syntax:

LINE [STEP][(<x1>, <y1>)] - [STEP] (<x2>, <y2>) [, <Zeichenfarbe>] [, B[F]] [, <Bitmuster>]

GW-Basic zeichnet eine Linie von Punkt (x1,y1) nach (x2,y2). Fehlt erstere Angabe, so nimmt der Interpreter den zuletzt gezeichneten Punkt (beziehungsweise die Bildschirmmitte) als Anfangskoordinate der Linie. Für die Zeichenfarbe gelten die bereits beim Pset-Befehl genannten Erklärungen.

Füllen Sie Rechtecke mit selbstdefinierten Mustern

Haben Sie dem Befehl zusätzlich die Option »B« angefügt, dann zeichnet GW-Basic keine Linie, sondern ein Rechteck (»box«), dessen Ecken durch die beiden angegebenen Koordinaten bestimmt sind. Zur genauen Definition eines Rechtecks genügen ja zwei Eckpunkte. Steht hinter der Option »B« noch ein »F« (»fill«), dann füllt der Interpreter das gezeichnete Rechteck in der aktuellen Zeichenfarbe.

Der ganzzahlige Parameter »Bitmuster« wird bitweise zerlegt. Die ersten 16 Bit dieses Werts bestimmen das Bitmuster, mit dem die Linie beziehungsweise das Rechteck auf dem Bildschirm erscheint. Ein gesetztes Bit entspricht einem gesetzten Bildpunkt und umgekehrt. Kreise und Ellipsen zeichnet GW-Basic durch den Circle-Befehl:

CIRCLE [STEP] (<x>, <y>), <Radius> [, <Zeichenfarbe>] [, <Ausschnitt1>, <Ausschnitt2>] [, <Achsenverhältnis>]

Die Koordinate (x,y) bestimmt den Mittelpunkt der zu zeichnenden Fi-

gur und der Radius gibt den halben Kreisdurchmesser an. Die Angaben »Ausschnitt1« und »Ausschnitt2« interpretiert GW-Basic als Winkel im Bogenmaß (-2xPi bis 2xPi) und zeichnet entsprechend nur den Kreis- beziehungsweise Ellipsenausschnitt, der durch diese beiden Winkel eingegrenzt ist. Das Achsenverhältnis legt die Form der Ellipse fest. Ein Achsenverhältnis von 1,0 (»1:1«, bezogen auf das Verhältnis von x- zu y-Achse) bestimmt einen Kreis. Ist der Wert größer als 1, erscheint die Ellipse in der y-Achse (vertikal) gestreckt, bei Zahlen kleiner 1 erfolgt die Streckung in der x-Achse (horizontal).

Dem Ausfüllen von Flächen dient der Paint-Befehl:

PAINT [STEP] (<x>, <y>)
[, <Modus>] [, <Randfarbe>]

GW-Basic beginnt mit dem Füllen des Grafikbildschirms bei der Koordinate (x,y). Begrenzt wird die Fläche durch eine Linie in der angegebenen Randfarbe, wobei hier Werte zwischen 0 und 3 (CGA) oder 0 und 15 (EGA oder VGA) zulässig sind. Wenn der Modus innerhalb dieser Bereiche liegt, dann füllt der Interpreter den Bereich mit der so angegebenen Farbnummer. Statt dessen können Sie auch ein Bitmuster angeben, das in der aktuellen Zeichenfarbe dann zum Füllen des Bereichs verwendet wird. Das Bitmuster teilen Sie dem Paint-Befehl über maximal 64 aneinandergehängte Chr\$-Anweisungen mit, also zum Beispiel

```
paint (100,200), chr$(170)+chr$(255)+chr$(15),3
```

GW-Basic interpretiert die Bitwerte der einzelnen Zahlen übereinandergelegt und bildet so das Füllmuster. Ein sehr leistungsfähiges, wenn auch umständlich zu programmierendes Grafikkommando ist »draw«. Die Schreibweise erscheint einfach:

DRAW <Kommandostring>

Der Kommandostring ist, wie es der Name bereits andeutet, eine Zeichenkette, die sowohl als String-Konstante als auch als Zeichenkettenvariable angegeben wird.

GW-Basic hat sogar einen computergesteuerten Zeichenstift

Innerhalb dieses Strings sind einige Zeichenbefehle zulässig, wobei der Zeichenursprung entweder beim zuletzt gesetzten Punkt (eventuell mit »pset« definieren) oder, falls nicht vorhanden, im Bildschirmmittelpunkt liegt. Die einzelnen Befehle folgen durch ein Leerzeichen getrennt aufeinander.

Wenn Sie anstatt eines der angegebenen Parameter ein Gleichheitszeichen und einen Variablennamen hinter den Befehl setzen, dann holt sich GW-Basic den Parameter aus der Variablen (Beispiel: »U = oben%«). Nachfolgend finden Sie eine Liste dieser Befehle:

u<n> (»up«, »nach oben«)

zeichnet eine Linie um »n« Punkte nach oben.

d<n> (»down«, »nach unten«)

zeichnet eine Linie um »n« Punkte nach unten.

l<n> (»left«, »nach links«)

zeichnet eine Linie um »n« Punkte nach links.

r<n> (»right«, »nach rechts«)

zeichnet eine Linie um »n« Punkte nach rechts.

e<n>

zeichnet eine diagonale Linie um »n« Punkte nach rechts oben.

f<n>

zeichnet eine diagonale Linie um »n« Punkte nach rechts unten.

g<n>

zeichnet eine diagonale Linie um »n« Punkte nach links unten.

h<n>

zeichnet eine diagonale Linie um »n« Punkte nach links oben.

b<Richtung>

verschiebt den Zeichenstift um einen Punkt in die angegebene Richtung, ohne jedoch Punkte zu setzen. Für die Richtung geben Sie einen Buchstaben aus der Reihe »UDLREFGH« entsprechend den oben genannten Bedeutungen an (also verschiebt zum Beispiel »BR« die Zeichenkoordinate um einen Punkt nach rechts).

n<Richtung>

Dies entspricht dem B-Befehl, wobei aber der Punkt diesmal gesetzt wird.

m<x>, <y> (»move«, »bewege«)

zeichnet eine Linie vom aktuellen Zeichenpunkt zu der angegebenen Koordinate. Wenn Sie zu den Parametern »x« und »y« ein Vorzeichen mit angeben, dann zeichnet der Interpreter vom aktuellen Punkt ausgehend, jeweils »x« und »y« Punkte weit in die angegebene Richtung. Zum Beispiel zeichnet »paint "m + 3, -10"« vom aktuellen Punkt ausgehend drei Bildpunkte nach rechts und zehn nach oben (der Koordinatenursprung (0,0) liegt in der linken oberen Ecke).

c<n> (»color«, »Farbe«)

legt eine neue Zeichenfarbe fest, wobei »n« eine Farbnummer aus der Palette darstellt.

p<f1>, <f2> (»paint«, »malen«)

füllt ab der aktuellen Zeichenposition einen Bereich, der durch Punkte

in der Farbe »f2« eingeschlossen ist, mit der Farbe »f1« (jeweils Nummern in der Farbpalette).

a<n> (»angle«, »Winkel«)

legt einen Umrechnungswinkel für nachfolgende Bewegungen fest. Dadurch bewegt der E-Befehl beispielsweise den Zeichenstift nicht mehr nach rechts oben, sondern nach links unten. Der Parameter »n« ist wie folgt zu interpretieren: 0 für 0 Grad, 1 für 90 Grad, 2 für 180 Grad und 3 für 270 Grad.

ta<n>

entspricht dem A-Befehl, jedoch können Sie den Winkel »n« zwischen -360 und 360 Grad frei wählen. Negative Werte entsprechen einer Verschiebung gegen den Uhrzeigersinn und positive Werte entsprechen einer Verschiebung im Uhrzeigersinn.

s<n>

definiert einen Vergrößerungsfaktor für alle anderen Zeichenbefehle. Die Zahl »n« wird von GW-Basic durch 4 geteilt. So bewegt der N-Befehl beispielsweise nach dem Kommando »S16« den Punkt nicht um eine, sondern um vier Koordinaten weiter.

x<Zeichenkettenvariable>

führt die in der angegebenen Zeichenkettenvariablen enthaltenen Befehle sofort aus. Das Dollarzeichen müssen Sie mit angeben (zum Beispiel »Xa\$«).

Die Programmierung von mathematischen Graphen ist leichter als Sie denken

Die nachfolgenden GW-Basic-Grafikbefehle dienen nicht dem Setzen oder Löschen von Grafikpunkten, sondern der Unterstützung eines entsprechenden Programms:

POINT(<x>, <y>)

Diese Funktion (»a = point(12,50)«) liefert die Farbnummer eines Punktes an der Stelle (x,y). Die Farbnummer bezieht sich dabei auf eine Position innerhalb der gerade gültigen Palette.

VIEW [SCREEN] (<x1>, <y1>) - (<x2>, <y2>) [, <Hintergrundfarbe>] [, <Rahmenfarbe>]

Mit dem View-Kommando setzen Sie innerhalb des gesamten, darstellbaren Bildschirmbereichs ein Bildschirmfenster in der angegebenen Hintergrundfarbe mit einem optionalen Rahmen in der Rahmenfarbe fest (durch die beiden Koordinaten (x1,y1) und (x2,y2) definiert), auf das sich alle anderen Grafikbefehle beziehen. Der Koordinatenursprung (0,0) liegt daraufhin in der linken oberen Ecke dieses Fensters. Beachten Sie bitte, daß sich die maximalen Werte

in x- und y-Richtung dann entsprechend vermindern.

Ist »screen« mit angegeben, dann beziehen sich alle Koordinaten auch weiterhin absolut auf die linke obere Ecke des Gesamtbildschirms. Werte, die außerhalb des definierten Fensters liegen, werden dann jedoch ignoriert.

WINDOW [SCREEN] (<x1>, <y1>) - (<x2>, <y2>)

Dieser Befehl erlaubt Ihnen, ein eigenes Koordinatenkreuz zu definieren. Das bedeutet zum einen, daß der Koordinatenursprung (0,0) nicht mehr links oben, sondern in der Bildschirmmitte liegt, und zum anderen, daß Sie die Schrittweite in x- und y-Richtung frei wählen können. Ein Beispiel: Sie benötigen für eine Sinuskurve einen Wertebereich von -10 bis 10 in der x-Achse und von -1 bis 1 in der y-Achse. Das Kommando

window (-10,-1) - (10,1)

programmiert die Koordinatenangaben von GW-Basic entsprechend um. Ab sofort sind für alle anderen Grafikbefehle nur noch Werte innerhalb dieser Grenzen sichtbar. Die Umrechnung mit entsprechender Rundung übernimmt der Interpreter automatisch. Ohne die Option »screen« werden die Angaben (x1,y1) und (x2,y2) als linke untere und rechte obere Ecke aufgenommen, mit »screen« interpretiert der Computer (x1,y1) als linke obere und (x2,y2) als rechte untere Ecke.

PMAP(<Koordinate>), <Art>

Diese Funktion dient der Umrechnung von Real- in mit »window« definierte Koordinaten und umgekehrt. Zuvor muß natürlich ein entsprechender Window-Befehl erfolgt sein. Der Parameter »Art« ist ein Wert zwischen 0 und 3 und gibt die Art der Umrechnung an:

0 - Die Koordinate wird als x-Koordinate innerhalb eines benutzerdefinierten Koordinatenkreuzes angesehen und in die entsprechende Real-Koordinate umgerechnet.

1 - Wie 0, jedoch für die y-Koordinate.

2 - Die Koordinate wird als reale x-Koordinate angesehen und in den entsprechenden Wert im Window umgerechnet.

3 - Wie 2, jedoch für die y-Koordinate.

GET [STEP] (<x1>, <y1>) - [STEP] (<x2>, <y2>), <Feldvariable>

Dieser Befehl liest einen durch die beiden Koordinaten (x1,y1) und (x2,y2) angegebenen, rechteckigen Bildschirmausschnitt in ein Variablenfeld ein. Das Variablenfeld muß mit Hilfe der Dim-Anweisung ent-

sprechend groß definiert sein, ein Integer-Feld ist zu empfehlen. Die Größe des Feldes in Bytes errechnet sich so:

int((<Breite>*<Faktor>+7)/8)*<Höhe>+4

»Breite« und »Höhe« sind entsprechend des zu speichernden Bildausschnitts festzulegen. Den Faktor setzen Sie auf 1 bei zweifarbigen, hochauflösenden Grafiken, auf 2 bei niedrigauflösenden, vierfarbigen Bildern und auf 4 bei EGA-Grafiken im hochauflösenden Modus. Bei 2000 Byte benötigen Sie so ein Variablenfeld mit 1000 Elementen, weil pro Element zwei Byte gespeichert werden können.

Auch Befehle zur Window-Programmierung hat GW-Basic in seinem Wortschatz

Für »Feldvariable« setzen Sie nur deren Namen ein, also zum Beispiel »a« für das Variablenfeld »a(...)«.

PUT (<x>, <y>), <Feldvariable>, <Modus>

schreibt einen mit »get« gespeicherten Bildschirmausschnitt ab der Koordinate (x,y) (linke, obere Ecke) wieder in den Grafikbildschirm. Die Art des Zurückschreibens geben Sie über den Parameter »Modus« an, der eines der folgenden Schlüsselwörter sein muß:

PSET - schreibt den Ausschnitt unverändert in das aktuelle Grafikbild
PRESET - invertiert den Ausschnitt vor dem Zurückschreiben.

XOR - verknüpft den gespeicherten Ausschnitt mit dem entsprechenden Bildschirminhalt über die logische XOR-Funktion:

0 xor 0 = 0
0 xor 1 = 1
1 xor 0 = 1
1 xor 1 = 0

AND - verknüpft den gespeicherten Ausschnitt mit dem entsprechenden Bildschirminhalt über die logische AND-Funktion:

0 and 0 = 0
0 and 1 = 0
1 and 0 = 0
1 and 1 = 1

OR - verknüpft den gespeicherten Ausschnitt mit dem entsprechenden Bildschirminhalt über die logische OR-Funktion:

0 or 0 = 0
0 or 1 = 1
1 or 0 = 1
1 or 1 = 1

Bei diesen Verknüpfungen entspricht eine 1 einem gesetzten und eine 0 einem nichtgesetzten Grafikpunkt.

(tr)

GW-Basic-Referenz: Sound

Tonerzeugung zählt nicht zu den Highlights der Hardware eines Personalcomputers. Dennoch kennt der GW-Basic-Interpreter einige leistungsfähige Befehle auf diesem Gebiet.

BEEP

Keine zusätzlichen Parameter sind erlaubt. »Beep« gibt einen 800-Hertz-Ton für die Dauer einer Viertelsekunde aus. Dies entspricht dem ASCII-Code 7 (»bell«), »print chr\$(7)«.

SOUND < Frequenz >, < Dauer >

Die Frequenz des auszugebenden Tons kann theoretisch zwischen 37 und 32767 Hertz liegen. Beachten Sie jedoch, daß der kleine PC-Lautsprecher nur einen schmalen Bereich aus diesem Frequenzspektrum wiedergeben kann. »Dauer« darf Werte zwischen 0 und 65535 annehmen und gibt die Anzahl der Prozessortakte an. Jeder Prozessortakt dauert eine Achtzehntelsekunde. Während ein Sound-Befehl ausgeführt wird, fährt GW-Basic weiter im Programm fort, bis es erneut auf einen Sound-Befehl trifft. Ist zu diesem Zeitpunkt die erste Tonfolge noch nicht ausgeklungen, wartet der Interpreter. Das MB-Kommando des Play-Befehls erlaubt jedoch auch beim Sound-Befehl Hintergrundverarbeitung über den 32 Zeichen großen Musikpuffer. Umfangreiche Melodien programmieren Sie mit dem Play-Befehl:

PLAY < Kommandostring >

Der Kommandostring stellt entweder eine Zeichenkettenkonstante oder eine Stringvariable dar. Die darin enthaltenen Kommandos führt GW-Basic nacheinander aus.

Buchstaben A bis G [+ , #] [-]

spielt einen Ton der jeweiligen Tonhöhe. Ein Plus- oder Nummernzeichen hinter der Note erhöht ihn um einen Halbton, ein Minuszeichen erniedrigt ihn um einen Halbton. Beachten Sie, daß die im deutschspra-

chigen Raum gebräuchliche Kennung »H« in GW-Basic als »B« bezeichnet wird.

L < n > (Länge)

Definiert die Länge aller folgenden Noten bis zum nächsten L-Kommando als Teile einer ganzen Note. »L8« legt beispielsweise Achtelnoten fest. Die Notenlänge kann aber auch direkt hinter die jeweilige Notenangabe gesetzt werden, wobei dieser Wert dann nur für diesen einen Ton gilt. So spielt zum Beispiel »C16« eine Sechzehntelnote in der Tonhöhe »C«.

. (Punktieren einer Note)

Verlängert die Ausführungszeit einer Note um die Hälfte. »G4.« spielt also eine punktierte Viertelnote der Länge drei Achtel.

T < n > (Tempo)

Legt das Abspieltempo Ihres Musikstücks fest. »n« spezifiziert die Anzahl der Viertelnoten pro Minute und muß im Bereich zwischen 32 und 255 liegen.

N < n > (Notennummer)

Dies ist eine Alternative, eine Note zu spielen. »n« gibt dabei die Nummer des jeweiligen Halbtons (1 bis 84) innerhalb der möglichen sieben Oktaven an (7 Oktaven mal 12 Halbtöne gleich 84 Notennummern). Bei »n« gleich 0 setzt GW-Basic eine Pause.

P < n > (Pause)

Hier setzt GW-Basic eine »n-tel« Pause. »P8« bewirkt also eine Pause von der Länge einer Achtelnote. Für »n« sind Werte zwischen 1 und 64 zulässig.

MF (mode foreground)

Das Spielen von Noten läuft im Vordergrund ab, was bedeutet, daß der Interpreter bei einer erneuten Play-Anweisung die Beendigung der er-

sten abwartet. Unterdessen fährt er mit der Programmausführung fort.

MB (mode background)

Alle angeforderten Noten schreibt GW-Basic in einen 32 Byte großen Musikpuffer, der im Hintergrund unter Zuhilfenahme des On-Play-Befehls bearbeitet wird. Während der Tonerzeugung läuft das Basic-Programm normal weiter.

ML (music legato)

Alle nachfolgenden Noten besitzen ihre volle, angegebene Länge.

MS (music staccato)

Die zu spielenden Noten haben nur noch drei Viertel ihrer definierten Länge.

MN (music normal)

Zu spielende Noten werden in sieben Achtel ihrer Länge ausgeführt. Dieser Modus ist die Standardeinstellung.

O < n > (Oktave)

Setzt die zu spielende Oktave für nachfolgende Ton-Kommandos. Werte zwischen 0 und 6 sind zulässig, die Standardeinstellung lautet hier 4.

>

Erhöht die gerade eingestellte Oktave um 1.

<

Erniedrigt die gerade eingestellte Oktave um 1.

X < Stringvariable >;

Führt an dieser Stelle die in der Stringvariablen enthaltenen Kommandos als Unteroutine aus. Ein Beispiel hierzu wäre »Xa\$«, falls »a\$« weitere Steuerbefehle enthält. Anstelle eines numerischen Parameters können Sie bei den beschriebenen Befehlen auch einen Variablenamen angeben, dem ein Gleichheitszeichen voran- und ein Semikolon nachgestellt ist (»T = tempo%;«).

ON (< Pufferminimum >) GOSUB < Zeile >

PLAY ON | OFF | STOP

Diese Befehle sind ausführlich im Beitrag »GW-Basic-Referenz: Event-Handling« dieser DOS-Extra-Ausgabe beschrieben.

(tr)

```
[NA5] 1 REM BOLERO.BAS, Sprache GW-Basic
[R47] 5 REM Demo-Programm für Hintergrundmusik
[SKA] 10 ON PLAY(2) GOSUB 1050 'Zeiger auf entsprechendes
      Unterprogramm
[ZU9] 20 PLAY ON 'Beobachtung des Hintergrundpuffers einschalten
[X56] 30 GOSUB 1000 'Die ersten Noten ausgeben
[LKB] 35 'Hier folgt das Hauptprogramm, das im Vordergrund
      und abgearbeitet wird
[ADK] 40 CLS:FOR A=1 TO 65536:LOCATE 10,10:PRINT "Zählerstand:";A:LOCATE 12,10:PRINT "Anzahl Noten im
      Puffer:";PLAY(0):NEXT A
[251] 50 STOP
[3D0] 60
[PS5] 1000 'Hier beginnt das Unterprogramm,
[YP5] 1010 'das bei Bedarf neue Noten in den
[2F5] 1020 'Hintergrundpuffer schreibt.
[PE5] 1050 READ A$: 'Data-Element einlesen
[TFG] 1060 IF A$="ende" THEN RESTORE 2060:GOTO 1050 'we-
      nn dies das letzte Element ist, dann von vorne be-
      ginnen.
```

```
[AM5] 1070 PLAY "mb"+A$ 'Noten in den Puffer
[GYC] 1080 RETURN 'dort fortfahren, wo die Programmaus-
      führung unterbrochen wurde
[VN0] 1090
[US6] 2000 'Hier beginnt der Datenbereich, in dem
[YE5] 2010 'sich die Notenwerte befinden.
[1T4] 2060 DATA "o2t8014>c18cl16<b>c<d>"
[TS5] 2070 DATA "c<ba18>c116c<a14>c18cl16<b>c"
[M05] 2080 DATA "<agef12g116gfedefgal4g"
[2G5] 2090 DATA "gl16gabagfeded18cl16cd18ef"
[AW5] 2100 DATA "l4dl2g18p814>d18dc116<bab>cd"
[6G6] 2110 DATA "cl8<bl16b>c<ba>c<baf18fl16ffl8fa"
[AD5] 2120 DATA "l16>c<abg18fl16ffl8fal16bg"
[995] 2130 DATA "af18dl16dc14dl8dl16dd18df"
[LS4] 2140 DATA "l16afgel8dl16dc14dl8d"
[7G5] 2150 DATA "l16dc18dl16ef12g116gfed13c"
[G52] 2160 DATA "ende"
```

Listing. »musik.bas« zeigt die Hintergrundfähigkeiten der GW-Basic-Sound-Befehle

GW-Basic-Referenz: Ein- und Ausgabe

Basic stellt drei grundlegende Gruppen von Befehlen für die Ein- und Ausgabe zur Verfügung: die Befehle für den Bildschirm, den Drucker und Dateien. Die nachfolgenden Beschreibungen sind in diese drei Unterbegriffe aufgeteilt. Steht ein Wort in spitzen Klammern, so steht an dieser Stelle ein Parameter. Ausdrücke in eckigen Klammern sind optional, sie müssen also nicht angegeben werden.

Eingaben von der Tastatur und Ausgaben auf den Bildschirm

COLOR [<Vordergrund>] [<Hintergrund>]

Festlegen je einer Farbe für den Bildschirmhintergrund und die Zeichensfarbe. Das Kommando kann auch nur mit dem ersten Parameter aufgerufen werden. Als Farben lassen sich für den Vordergrund alle 16 und für den Hintergrund acht Farben aus der in Tabelle 1 abgedruckten Palette auswählen. Die Farben sind nur bei einem Farbgrafikadapter ansprechbar.

```
10 color 10,3
```

CLS

Löschen des Bildschirms. Bei Farbmonitoren oder Farbgrafikkarten werden dabei die mit dem Color-Befehl gesetzten Farben für Vorder- und Hintergrund verwendet.

```
10 cls
```

INKEY\$

Auslesen einer einzigen gedrückten Taste. Diese Funktion ist nützlich, wenn lediglich ein Zeichen von der Tastatur eingelesen werden soll. Das Zeichen wird nicht auf dem Bild-

schirm dargestellt. Handelt es sich bei der betätigten Taste um eine Steuertaste (zum Beispiel [Page Up] oder [F1]), so ist die zurückgegebene Zeichenkette zwei Zeichen lang. Das erste Zeichen wird dabei auf ASCII 0 gesetzt, um den Steuercode anzuzeigen. Damit kann »inkey\$« auch alle erweiterten Tastencodes verarbeiten. Wenn von der Funktion ein Leerstring zurückgegeben wird (nicht chr\$(0)!), so wurde zum Zeitpunkt der Abfrage keine Taste betätigt.

```
10 while inkey$ = "" : wend
```

INPUT [;] ["<Aufforderung>"] [;].<Variable> ...

Eingabe von Daten über die Tastatur. Hinter dem optionalen Text <Aufforderung> wird die Eingabe eines Textes, einer Zahl oder eines gemischten Ausdrucks verlangt. Der Anwender muß die Eingabe mit der Taste [Enter] abschließen. Ein Semikolon direkt hinter dem Input-Befehl unterdrückt hier das Weiterrücken des Cursors in die nächste Zeile. Die eingegebenen Daten werden in die durch <Variable> angegebene Variable übernommen.

Das Fragezeichen des Interpreters kann unterdrückt werden, wenn der Variablenname mit einem Semikolon vom Ausgabertext getrennt wird. Soll kein Text auf dem Bildschirm erscheinen, dann sind hier zwei direkt aufeinanderfolgende Anführungszeichen erforderlich. Werden mehrere Variablen durch Kommata getrennt angegeben, so muß der Anwender für jede Variable eigene Daten eingeben, die er wiederum durch Kommata oder [Enter] trennt.

```
10 input "Wie ist ihr Name";nname$
20 input "Drei Zahlen eingeben: ",z1%,z2%,z3%
30 print "Noch eine Zahl: "; : input "",a
```

INPUT\$(<Zeichenanzahl>)

Einlesen einer festgelegten Anzahl von Zeichen von der Tastatur. Die Eingabe wird nicht durch [Enter] beendet, sondern endet dann, wenn alle durch »Zeichenanzahl« festgelegten Zeichen eingegeben oder die Funktion durch die Tastenkombination [Ctrl-C] abgebrochen wurde. Die

eingegebenen Zeichen werden nicht auf dem Bildschirm dargestellt.

```
10 eingabe$=input$(10)
20 a=100:e$=input$(a)
```

LINE INPUT [;] ["<Aufforderung>"] [;].<Zeichenkettenvariable>

Mit diesem Kommando werden auch Sonderzeichen in einer Eingabe von der Tastatur verarbeitet. So kann in der eingegebenen Zeichenkette zum Beispiel auch ein Komma oder ein Anführungszeichen enthalten sein. Der Line-Input-Befehl akzeptiert nur Zeichenkettenvariablen zum Speichern der gelesenen Daten.

```
10 line input "Geben Sie ruhig ein Komma mit ein: ",tt$
20 line input zkette$
```

LOCATE [<Zeile>] [, <Spalte>] [, <Zustand>] [, <Start>] [, <Ende>]

Setzen des Cursors auf eine neue Position für die nächste Print-Anweisung. Mit <Zeile> wird die neue Bildschirmzeile des Cursors festgelegt, <Spalte> wählt die neue Bildschirmspalte aus. Die zulässigen Werte für beide Parameter sind abhängig vom gewählten Videomodus. <Zustand> gibt an, ob der Cursor sichtbar sein soll oder nicht. Ein Wert von 0 schaltet den Cursor aus, ein Wert von 1 schaltet ihn ein. Mit den Parametern <Start> und <Ende> kann die Größe des Cursors verändert werden. Mit <Start> wird die Nummer der Rasterzeile innerhalb eines darstellbaren Zeichens angegeben, an der die Darstellung des Cursors beginnen soll, mit <Ende> wird die Nummer der Rasterzeile ausgewählt, an der die Darstellung des Cursors enden soll. Die erlaubten Werte für <Start> und <Ende> hängen von der verwendeten Grafikkarte ab. Je nach Anzahl der Punkte eines Zeichens lassen sich hier unterschiedliche Ergebnisse erzielen.

```
10 locate 1,20 : print "Hallo 1"
20 locate 1,1,0,0,2 : print "Hallo 2"
```

PCOPY <Quellseite>, <Zielseite>

Grafikkarten, die über mehrere einzeln ansprechbare Bildschirmseiten verfügen, werden mit diesem Befehl

Farbwert	Farbe
0	schwarz
1	blau
2	grün
3	türkis
4	rot
5	purpur
6	braun
7	hellgrau
8	dunkelgrau
9	hellblau
10	hellgrün
11	helltürkis
12	hellrot
13	hellpurpur
14	gelb
15	weiß

Tabelle 1. Die Farbwerte für den Color-Befehl

Hercules	unter GW-Basic nicht ansprechbar
MDA	eine
CGA	vier im 80-Zeichen-Modus
CGA	acht im 40-Zeichen-Modus
EGA	bis zu acht Seiten (je nach Speicherausbau)
VGA	acht Seiten

Tabelle 2. Anzahl der Bildschirmseiten bei den verschiedenen Grafikkarten

veranlaßt, Bildschirminhalte von einer Seite zu einer anderen zu kopieren. Der erste Parameter gibt die Seite an, die in die durch den zweiten Parameter festgelegte Seite kopiert werden soll. Tabelle 2 gibt Aufschluß über die verfügbaren Bildschirmseiten je nach Grafikkarte. Die Seitennummern beginnen in der Zählung für den Pcopy-Befehl bei 0.

```
10 pcopy 0,1
```

POS(<Dummy-Argument>)

Auslesen der aktuellen Cursorspalte (X-Position) auf dem Bildschirm. Das <Dummy-Argument> dient nur zur korrekten Syntax des Befehls und kann einen beliebigen Wert annehmen. Normalerweise benutzt man hier 0. Der zurückgelieferte Wert liegt zwischen 1 und 40 oder zwischen 1 und 80.

```
10 if pos(0)=10 then print "Cursor in Spalte 10!"
```

CRSLIN

Feststellen der aktuellen Cursorzeile (Y-Position) für die nächste Print-Anweisung. Zurückgeliefert wird ein Wert zwischen 1 und 25.

```
10 if crslin=1 then locate 25,1
20 if crslin=10 then print "Bildschirmzeile 10 erreicht"
```

PRINT [<Ausdruck>] [,] [:] ...

Der <Ausdruck>, der mit dem Print-Kommando auf dem Bildschirm ausgegeben wird, kann eine Konstante beliebigen Typs (also Ganzzahl, einfachgenaue Fließkommazahl, doppeltgenaue Fließkommazahl oder Zeichenkette), eine Variable beliebigen Typs, eine Funktion oder ein beliebig zusammengesetzter Ausdruck aus Konstanten, Variablen und Funktionen sein. Als Zeichenfarbe wird die mit dem Color-Befehl angegebene Zeichenfarbe benutzt. Sind die Ausdrücke durch Kommata getrennt, rückt der Cursor bei der Ausgabe jeweils um eine Tabulatorposition weiter. Ein Semikolon verändert die Cursorposition nicht und verhindert auch das Ausgeben eines Return-Zeichens (ASCII-Wert 13) nach dem Print-Befehl. Als Trennzeichen zwischen Ausdrücken kann es meist weggelassen werden. Ist der Ausdruck ein numerischer Wert, so fügt der Interpreter automatisch ein Leerzeichen vor und nach der Ausgabe ein.

```
10 print "Der Wert der Formel
100+2*30-3^2 ist ";100+2*30-3^2
20 print fn f(100)*100-1.234*a1%*20
30 print "Werte:" a b
c,"Strings:" a$ und "b$c$
```

PRINT USING <Maske> ... ; <Ausdruck> ...

Für die Darstellung von Werten in einem bestimmten Format wird im Print-Befehl hinter dem Schlüsselwort »using« mit <Maske> eine Zei-

chenkette angegeben, die das Format des nach dem folgenden Semikolon stehenden Ausdrucks für die Ausgabe bestimmt. Die folgenden Zeichen sind in der Maske erlaubt:

- # – Dezimalstelle reservieren
- , – Tausendertrennung vornehmen
- ** – Sternchen für freie Stellen einfügen
- \ – Anzahl der Zeichen einer Zeichenkette festlegen
- ! – erstes Zeichen einer Zeichenkette auswählen
- . – Position der Dezimalpunkte festlegen
- ^ – Exponentialstelle reservieren
- \$\$ – Dollarzeichen setzen
- & – keine Formatierung von Zeichenketten vornehmen

In der Maske wird mit dem Zeichen »#« festgelegt, wo Zahlen im Ausdruck erscheinen. Das Zeichen »#« darf mit einem Dezimalpunkt kombiniert werden. Wünschen Sie zum Beispiel Dezimalzahlen mit drei Vorkomma- und zwei Nachkommastellen, so ist das mit der Maske »##.#.« zu erreichen. Mit einem Pluszeichen am Anfang oder Ende der Maske wird das Vorzeichen der Zahl an der entsprechenden Stelle ausgegeben.

Ein Minuszeichen am Ende der Maske weist den Interpreter an, das Vorzeichen hinter der Zahl auszugeben, wenn die Zahl negativ ist. Mit dem Exponentialzeichen »^« lassen sich Stellen für die Exponenten in der wissenschaftlichen Schreibweise von Zahlen reservieren. Mit einem Komma in einer Maske wird veranlaßt, daß jeweils drei Ziffern zusammengefaßt und von den folgenden Ziffern durch ein Komma abgesetzt werden. Zwei Sternchen zu Beginn einer Maske weisen den Interpreter an, unbenutzte Stellen der Maske mit Sternchen auszufüllen. Wird in einer Maske ein Dollarzeichen verwendet, so wird bei der Ausgabe ein Dollarzeichen vor oder hinter den Wert gesetzt.

Zur Formatierung von Zeichenketten dienen die Maskensymbole »\«, »!« und »&«. Mit »\« wird die Anzahl der darzustellenden Zeichen einer Zeichenkette festgelegt. Es werden so viele Zeichen dargestellt, wie durch die Anzahl der zwischen zwei »\« stehenden Zeichen (inklusive beider »\«) angegeben ist. So werden mit der Maske

```
\[hier stehen drei Leerzeichen]\
```

fünf Zeichen einer Zeichenkette ausgegeben. Das Ausrufezeichen bewirkt, daß nur das erste Zeichen einer Zeichenkette ausgegeben wird. Das Zeichen »&« sorgt dafür, daß die Zeichenkette unformatiert ausgegeben wird, also ohne Verwendung einer Maske.

Sie können die Maske auch durch eine Zeichenkettenvariable angeben und mehrere Masken in einem einzigen Print-Using-Befehl verwenden.

```
10 print using "###.##";100
20 a$="###.##" : print using
a$,"###.##";22,44
```

SCREEN [<Modus>] [, <Farbe>] [, <Zielseite>] [, <Zeigseite>]

Bei der Verwendung unterschiedlicher Bildschirmseiten bietet Basic die Möglichkeit, eine andere Seite darzustellen als diejenige, auf der die Ausgaben eines Programms erfolgen. So kann man mit Print-Befehlen eine Bildschirmseite aufbauen, während eine andere dargestellt wird. Mit <Modus> wird der Bildschirmmodus der Grafikkarte gewählt (siehe Artikel »Die Kartenfrage« in dieser DOS-Extra-Ausgabe). <Farbe> legt fest, ob der verwendete Bildschirm farbfähig ist oder nicht. Ein Wert von 0 weist auf einen nicht farbfähigen Bildschirm hin, während ein Wert von 1 eine Farbgrafikkarte anzeigt. Mit <Zielseite> wird festgelegt, auf welche der Bildschirmseiten sich die normalen Ausgaben wie »print«, »color« und so weiter auswirken sollen. <Zeigseite> schließlich wählt aus, welche Seite der Grafikkarte auf dem Bildschirm sichtbar ist. Die Angaben von <Zeigseite> und <Zielseite> sind optional.

```
10 screen 1
20 screen 0,0,0,1
```

SPC(<Zeichenanzahl>)

Einfügen einer mit <Zeichenanzahl> vorgegebenen Anzahl von Zeichen in die aktuelle Ausgabezeile. Falls die angegebene Zeichenanzahl nicht mehr in der aktuellen Bildschirmzeile unterzubringen ist, wird die Ausgabe der Freizeichen in der nächsten Zeile fortgesetzt.

```
10 print "erster Text" spc(20) "zweiter
Text" spc(10) "dritter Text"
20 a$="zehn Zeichen" : print
spc(79-len(a$));a$
```

TAB(<Spalte>)

Diese Funktion erlaubt es, den Cursor innerhalb einer Zeile zu positionieren. <Spalte> enthält die Nummer einer Bildschirmspalte. Falls der Wert von <Spalte> kleiner als die aktuelle Cursorspalte ist, wird die entsprechende Spalte in der nächsten Zeile angesteuert.

```
10 print tab(5); "Text beginnt bei Spalte 5"
20 print 100; tab(10); 200; tab(20); 300n
```

VIEW PRINT [<Startzeile> TO <Endzeile>]

Einrichten eines virtuellen Bildschirms. Der durch die Zeilen zwischen den Parametern <Startzeile> und <Endzeile> festgelegte Bereich wird als neuer Gesamtbildschirm betrachtet. Alle bildschirmorientierten Befehle (wie »cls«, »print« und so

weiter) beziehen sich nach dem Kommando auf den neuen, virtuellen Bildschirm. Auch Befehle zur Cursor-Positionierung (wie »locate«, »crsln« und so weiter) werden nach »view print« als relativ zum Beginn des neuen Textfensters betrachtet (»locate 1,1« setzt den Cursor dann in die linke obere Ecke des Bildschirmfensters, nicht in die Ecke des gesamten Bildschirms). Durch Verwendung des Befehls ohne Parameter wird der normale Bildschirm mit den gewohnten Einstellungen wieder wirksam.

```
10 view print 10 to 20
20 view print
```

WRITE <Ausdruck> [;] ...

Oft kommt es vor, daß man Zeichenketten in Anführungszeichen auf dem Bildschirm ausgeben möchte. Dazu dient der Write-Befehl. Er gibt alle Ausdrücke in Anführungsstrichen und durch Kommata getrennt auf dem Bildschirm aus.

```
10 write "testtext";100
20 a$="100" : write a$+"00";10000
```

Druckerausgabe

LLIST [<Startzeile>] [-] [<Endzeile>]

Ausdrucken eines Basic-Programmlistings auf einem angeschlossenen Drucker. Um nur bestimmte Zeilen des Listings zu drucken, werden mit den beiden Parametern <Startzeile> und <Endzeile> Anfang und Ende dieses Bereiches angegeben. Das erste »l« von »llist« kommt vom englischen Wort für Zeilendrucker: »line printer«.

```
llist
llist 100-500
```

LPRINT [<Ausdruck>] [,] [;] ...

Ausgabe von Ausdrücken über einen an der Parallelschnittstelle angeschlossenen Drucker. Es gelten die gleichen Vereinbarungen wie beim normalen Print-Befehl.

```
10 lprint "Druckertest Nummer";1
```

LPRINT USING <Maske> ... ;<Ausdruck> ...

Ausgabe von Ausdrücken über einen an der Parallelschnittstelle angeschlossenen Drucker. Es gelten die gleichen Vereinbarungen wie beim normalen Print-Using-Befehl.

```
10 lprint using "##.##.##.##";100
20 a$="##.##.##.##" : lprint using a$,"##.##.##.##";22,44
```

Ein- und Ausgabe von und in Dateien

CLOSE # [<Dateinummer>] [, <Dateinummer>] ..

Schließen einer durch <Dateinummer> spezifizierten geöffneten Datei. Der Close-Befehl sorgt automatisch dafür, daß alle möglicherweise noch im Datenpuffer stehenden Daten auf

Diskette oder Festplatte geschrieben werden. »Close« ohne eine Dateinummer schließt alle offenen Dateien.

```
10 open "test.txt" for output as #1
20 close #1
```

FIELD # <Dateinummer> , <Länge1> AS <Feld1> [, <Länge2> AS <Feld2>] ...

Reservierung bestimmter Feldnamen für Teile des Datensatzbereiches einer relativen Datei. Damit lassen sich symbolische Feldnamen erzeugen, mit denen der Datensatzpuffer des Interpreters adressiert werden kann. Der in <Länge> zu jeder Feldvariablen <Feld> angegebene Wert reserviert die angegebene Anzahl an Bytes für die Variable. Feldvariablen müssen Zeichenkettenvariablen sein. Die Feldvariablen bleiben nur solange gültig und adressierbar, wie die relative Datei nicht geschlossen wurde.

```
10 field #1 , 25 as na$, 20 as vor$, 30 as be$, 30 as st$, 10 as te$
20 field #5 , 2 as erstzwei$, 2 as zweitzwei$
```

EOF(<Dateinummer>)

Überprüfung, ob das Dateiende der durch <Dateinummer> gewählten Datei erreicht ist. Die Funktion liefert den Wert -1 (logisch wahr), wenn das Ende der Datei erreicht ist, ansonsten den Wert 0 (logisch unwahr).

```
10 if not eof(1) then print "kein Dateiende"
20 a%=eof(5)
```

GET # <Dateinummer> [, <Satznummer>]

Auslesen eines Datensatzes einer relativen Datei. In <Dateinummer> wird die Nummer der relativen Datei angegeben, während <Satznummer> die Nummer des zu lesenden Datensatzes enthält. Wird keine <Satznummer> angegeben, so liest Basic den durch den Dateizeiger festgelegten aktuellen Datensatz und erhöht danach den Dateizeiger um den Wert 1. So kann mit mehreren sukzessiven Get-Kommandos sequentiell aus einer relativen Datei gelesen werden.

```
10 open "test.tst" as #1
20 get #1,100
30 get #1
```

INPUT # <Dateinummer> , <Variable> [, <Variable>] ...

Einlesen von Ausdrücken aus einer geöffneten Datei. Es gelten die gleichen Vereinbarungen wie beim normalen Input-Befehl, allerdings muß hinter dem Zeichen »#« die Nummer einer geöffneten sequentiellen Datei angegeben werden.

```
10 open "test.tst" for input as #1
20 input #1, nname$
30 input #1, dat1$, dat2$, dat3$
```

INPUT\$(<Zeichenanzahl> , <Dateinummer>)

Einlesen von Ausdrücken einer vor-

gegebenen Länge aus einer geöffneten sequentiellen Datei. Es gelten die gleichen Vereinbarungen wie beim normalen Input\$-Befehl, allerdings muß hinter dem Zeichen # die Nummer einer geöffneten Datei angegeben werden. Der Befehl ist vor allem dann nützlich, wenn Dateien zu bearbeiten sind, die sich mit dem Input-Befehl nicht mehr ordnungsgemäß einlesen lassen.

```
10 open "test.tst" for input as #1
20 a$=input$(19, #1)
30 print input$(10, #1)
```

LINE INPUT # <Dateinummer> , <Variable>

Einlesen von Ausdrücken aus einer geöffneten sequentiellen Datei, wobei in den Ausdrücken auch Trennzeichen, wie zum Beispiel Kommata und Anführungsstriche, enthalten sein dürfen. Es gelten die gleichen Vereinbarungen wie beim normalen Line-Input-Befehl, allerdings muß hinter dem Zeichen »#« die Nummer einer geöffneten sequentiellen Datei angegeben werden.

```
10 open "test.tst" for input as #1
20 line input #1, text1$
30 line input #1, a$
```

LOC(<Dateinummer>)

Ermittelt den aktuellen Wert des Dateizeigers einer relativen Datei. Als Parameter muß die Nummer der Datei in <Dateinummer> übergeben werden. Der Wert zeigt an, welchen Datensatz der Interpreter beim nächsten Put- oder Get-Kommando ohne Angabe einer Satznummer bearbeitet.

```
10 a=loc(1)
20 if loc(5)=200 then 1000
```

LOF(<Dateinummer>)

Gibt die Länge einer durch <Dateinummer> gewählten geöffneten Datei in Bytes aus. Dabei kann es sich um eine sequentielle oder eine relative Datei handeln.

```
10 la%=lof(1)
20 if loc(5) >= 100000 then print "Achtung: Diskette ist bald voll!"
```

LSET <Feld> = <Ausdruck>

Linksbündige Zuordnung eines Ausdrucks zu einer Feldvariablen, die mit dem Field-Kommando definiert wurde. Dabei werden Leerzeichen automatisch ergänzt und die Feldvariable so auf die festgelegte Länge gesetzt.

```
10 lset nachname$="Unbekannt"
20 lset es$="100"
```

OPEN "<Dateiname>" [for <Modus>] as # <Dateinummer> [len = <Satzlänge>]

Öffnen einer sequentiellen oder relativen Datei für die Ein- oder Ausgabe: <Dateiname> besteht aus maximal acht Zeichen des Dateinamens und einer maximal dreistelligen Namens-erweiterung. Der Modus gibt an, für

welche Operationen die Datei verwendet werden soll. Ist Modus nicht angegeben, öffnet GW-Basic eine relative, ansonsten eine sequentielle Datei. Es stehen die Modi »output«, »input« und »append« zur Verfügung. Damit lassen sich Datensätze in eine Datei schreiben (output), aus einer Datei lesen (input) oder an eine bestehende Datei anhängen (append). Die Dateinummer dient zur Auswahl der Datei bei späteren Lese- oder Schreibzugriffen, wobei Werte zwischen 1 und 15 zulässig sind. Der letzte Parameter schließlich ist optional, das heißt, er muß nicht mit angegeben werden. Er legt die Satzlänge einer relativen Datei fest. Hier ist ein Wert zwischen 1 und 32767 erlaubt. Ist der Len-Parameter im Open-Befehl angegeben, entfällt die Angabe des Modus für den Schreibzugriff, da bei relativen Dateien sowohl geschrieben als auch gelesen werden darf. Entweder öffnet man eine Datei also als sequentielle Datei, dann entfällt »len«, oder man öffnet sie als relative Datei, dann entfällt »for...as«.

Es gibt noch eine weitere, allerdings etwas weniger gebräuchliche Schreibweise des Open-Befehls:

```
open "<Modus>", #<Dateinummer>,
"<Dateiname>" [, <Satzlänge>]
<Dateinummer>, <Dateiname>
und <Satzlänge> entsprechen den
```

Parametern der zuerst genannten Schreibweise. <Modus> ist ein einzelner Buchstabe und wie folgt aufgeschlüsselt: »i« für »input«, »o« für »output«, »a« für »append« und »r« für relative Dateien (»random«).

```
10 open "daten1.dat" for output as #2
20 open "daten2.dat" as #2, len=100:
30 open "i", #3, "daten3.dat"
```

PRINT # <Dateinummer>, <Ausdruck> [,] [,] [<Ausdruck>] ..
Ausgabe von Ausdrücken in eine geöffnete Datei. Es gelten die gleichen Vereinbarungen wie beim normalen Print-Befehl, allerdings muß hinter dem Zeichen »#« die Nummer einer geöffneten sequentiellen Datei angegeben werden.

```
10 print #1, text$
20 print #1, "Ab in die Datei, Du";0
```

PUT # <Dateinummer> [, <Satznummer>]

Schreiben eines Datensatzes in eine relative Datei. Mit <Dateinummer> wird die Nummer der relativen Datei angegeben, während <Satznummer> die Nummer des zu lesenden Datensatzes enthält. Ist keine <Satznummer> angegeben, so schreibt der Interpreter in den aktuellen Datensatz (durch den Dateizeiger angegeben) und erhöht danach den Dateizeiger

um den Wert 1. So kann mit mehreren aufeinanderfolgenden Put-Kommandos sequentiell in eine relative Datei geschrieben werden.

```
10 put #1,100
20 put #1
```

RSET <Feld> = <Ausdruck>
Rechtsbündige Zuordnung eines Ausdrucks zu einer Feldvariablen, die mit dem Field-Kommando definiert wurde. Dabei ergänzt Basic automatisch Leerzeichen und setzt die Feldvariable so auf die festgelegte Länge.

```
10 rset beruf$="Angler"
20 rset al$=na$
```

WRITE # <Dateinummer>, <Ausdruck> [,] [,] [<Ausdruck>] ...

Ausgabe von Ausdrücken in eine geöffnete Datei, wobei alle Ausdrücke durch Kommata getrennt und in Anführungszeichen ausgegeben werden. Es gelten die gleichen Vereinbarungen wie beim normalen Write-Befehl, allerdings muß hinter dem Zeichen »#« die Nummer einer geöffneten sequentiellen Datei angegeben werden.

```
10 write #1, "Text1"; "Text2"
20 write #3, a$; "100"
```

(Martin Althaus/tr)

```
[9E6] 10 'Kopieren einer Datei in eine andere:
[HA5] 20 'Demonstration der Befehle zur
[036] 30 'Dateiverwaltung. Dabei werden alle
[PP6] 35 'Steuerzeichen aus der Datei entfernt.
[CU0] 40 CLS
[PT1] 50 ZAEHLER=1
[F57] 60 INPUT "Dateiname der Quelldatei ";QUELLE$
[KF6] 70 INPUT "Dateiname der Zieldatei ";ZIEL$
[3A3] 80 VIEW PRINT 10 TO 20
[W04] 90 OPEN QUELLE$ AS #1 LEN=1
[324] 100 FIELD#1,1 AS QUELLBYTES$
[MN4] 110 OPEN ZIEL$ FOR OUTPUT AS #2
```

```
[0S3] 120 WHILE ZAEHLER<LOF(1)
[D81] 130 GET#1
[CD3] 140 ZAEHLER=ZAEHLER+1
[624] 150 IF ASC(QUELLBYTES$)<32 THEN 180
[NN3] 160 PRINT #2,QUELLBYTES$;
[CE3] 170 PRINT QUELLBYTES$;
[371] 180 WEND
[GB1] 190 CLOSE
[Z82] 200 VIEW PRINT
[V21] 210 END
```

Listing. »Basdem« demonstriert einiger Dateibefehle zur Ein- und Ausgabe

Utilities gesucht

Programme, die den Bildschirminhalt auf den Kopf stellen, suchen unsere Leser sicherlich nicht. Aber einen wissenschaftlichen Taschenrechner, ein Festplatten-Backup-Programm, einen Extended-Memory-Treiber, Druckerspooles oder Cache-Speicher. Zögern Sie deshalb nicht, schicken Sie uns unverbindlich die aktuelle Version Ihres Programms und einen Ausdruck der Dokumentation.

Falls Sie sich vorweg informieren wollen, zwischen 10.00 und 12.00 Uhr und zwischen 15.00 und 17.00 Uhr stehen wir Ihnen an allen Werktagen unter der Rufnummer (089) 4 61 00 10 zur Verfügung.

Schicken Sie Ihre Unterlagen an folgende Adresse:
Redaktionsbüro Everts & Hagedorn
»DOS Software«
Hans-Stießberger-Str. 3
8013 Haar/München

GW-Basic-Referenz: Variablen

Ohne Variablen läuft in einem Computer gar nichts. Auch GW-Basic kennt verschiedene Typen dieser Platzhalter. Nur damit lassen sich Programme schreiben, die mit unterschiedlichen Daten arbeiten.

Variablen sind Speicherplätze, die innerhalb von GW-Basic zugewiesene Werte aufnehmen. Diese Werte können sowohl Zahlen als auch einzelne Zeichen oder bis zu 255 Byte lange Zeichenketten sein. Die Zuweisung erfolgt mit dem Befehl »let« oder mit einfacher Gleichsetzung. Die Programmzeilen

```
10 let a=10
20 a=10
```

sind beide korrekt und haben dieselbe Wirkung. Das Programm weist an dieser Stelle der Variablen »a« den Wert 10 zu. Nach dem Programmstart mit dem Run-Befehl haben alle Variablen den Wert 0, ein »goto <erste Programmzeile>« übernimmt die noch im Speicher befindlichen Werte.

Zahlenvariable können ganzzahlige Integer-Variablen sein oder einfachbeziehungsweise doppeltgenaue Zahlenwerte repräsentieren. Den Variablentyp erkennt GW-Basic an den Zusätzen, die der Name der Variablen trägt (siehe Tabelle).

Ein Array (Variablenfeld) ist eine Menge oder Tabelle mit gleichen Variablenamen. Jedes Element wird durch eine Nummer oder, je nach Dimensionierung, auch mehrere Nummern (Index) adressiert. So ist zum Beispiel »test(10)« Element eines eindimensionalen und »versuch\$(2,5)« Element eines zweidimensionalen Array. Maximal sind 255 Dimensionierungen mit höchstens 32767 Elementen zulässig. Eine Dimensionierung erfolgt mit dem Befehl »dim«. Ein Programm, das auf den Befehl »dim abc(20,50)« trifft, reserviert den Speicherplatz für das zweidimensionale Feld mit dem Namen »abc« und definiert die maximale Anzahl der Felder.

Der Befehl »erase« löscht Felder und erlaubt somit eine erneute Dimensionierung mit dem Dim-Kommando, was normalerweise nicht zulässig ist. »Erase abc« entfernt zum Beispiel das Variablenfeld »abc(...)« vollständig und gibt den reservierten Speicherplatz wieder frei.

Wenn ein Array mit dem Dim-Befehl nicht explizit dimensioniert wird, trägt die maximale Elementmenge zehn Variablen. Der kleinstmögliche Wert für den Index ist 0, solange dies

nicht mit dem Befehl »option base« geändert wurde. Trifft der Interpreter auf den Befehl »option base 1«, dann liegt der niedrigste Index bei 1. Andere Angaben als 0 (Voreinstellung) und 1 sind nicht zulässig.

Der Speicherplatzbedarf der verschiedenen Variablen ist unterschiedlich. Zeichenvariablen benötigen fünf Byte für ihre Verwaltung plus die Anzahl der enthaltenen Zeichen. Integer-Variablen brauchen zwei Byte, einfache Zahlen belegen vier Byte und doppeltgenaue Zahlenvariablen acht Byte Speicherplatz. Bei der Dimensionierung von Arrays reserviert GW-Basic für jedes Element den entsprechenden Speicherplatz.

Der Befehl »clear« setzt alle Zahlenvariablen auf den Wert 0, alle Zeichenvariablen auf die Länge 0 und schließt alle offenen Dateikanäle.

Wenn Sie ein Programm mit »run« starten, führt der GW-Basic-Interpreter automatisch zuerst diesen Befehl aus.

Falls notwendig, kann GW-Basic den Typ einer Variablen auch in einen anderen verwandeln. Näheres zu Typumwandlungen finden Sie im Artikel »GW-Basic-Referenz: Arithmetik«.

Besonders umfangreich ist in GW-Basic der Befehlssatz zur Verarbeitung von Zeichenketten. Als Beispiel wird hier die Zeichenkette »Das internationale Computermagazin« verwendet:

```
10 test$="Das internationale Computermagazin"
```

Dieser String lässt sich nun in einzelne Bestandteile zerlegen, so daß am Ende beispielsweise drei Zeichenketten mit dem Inhalt »Das«, »internationale« und »Computermagazin« bestehen.

```
20 test1$=left$(test$,3)
30 test2$=mid$(test$,5,14)
40 test3$=right$(test$,15)
```

In Zeile 20 weist das Programm der Variablen »test1\$« die ersten drei Zeichen des Strings »test\$« zu. Beachten Sie bitte, daß die »1« aus »test1\$« keinen Index einer Feldvariablen darstellt, sondern als Teil des Variablennamens zu sehen ist. In Zeile 30 erhält die Variable »test2\$« eine Zeichenkette, die an der fünften Stelle von »test\$« beginnt und 14 Zeichen lang ist. In Zeile 40 erfolgt eine Zuweisung der ersten 15 Zeichen von rechts aus »test\$« an die Variable »test3\$«.

Es besteht auch die Möglichkeit, eine Zeichenkette innerhalb einer anderen Zeichenkette zu suchen. Hierzu dient die Funktion »instr«.

```
50 print instr(5,test$,"Computer")
```

In dieser Zeile sucht das Programm den String »Computer« in der Zeichenkette »test\$«, wobei die Suche an der fünften Stelle beginnt. Das Programm gibt auf dem Bildschirm die Zahl 20 aus, da der String »Computer« ab der 20. Stelle von »test\$« steht.

Zum Feststellen der Länge einer Zeichenvariablen dient der Len-Befehl:

```
60 print len(test$)
```

Hier gibt der Computer die Länge der Zeichenkette »test\$«, also 35, auf dem Bildschirm aus.

Es besteht auch die Möglichkeit, eine Zeichenkette in eine Zahlenvariable zu verwandeln, wenn die Zeichenkette aus Zahlen besteht. Dies ist notwendig, wenn mit innerhalb von Strings enthaltenen Zahlen gerechnet werden soll, was normalerweise nicht möglich ist. So ergibt die Zeile

```
70 a$="1234 test" : test%=val(a$)
```

eine Zuweisung des Wertes 1234 an die Integer-Variable »test%«. Die Funktion »val« wandelt eine Zeichenkette in eine Zahl um, bis sie innerhalb dieser Zeichenkette auf ein Zeichen trifft, das nicht zu einer Zahl gehört (im Beispiel das »t« des Worts »test«). Umgekehrt können Sie mit der Funktion »str\$« eine Zahl in eine Zeichenkette verwandeln:

```
80 zeichen$=str$(test%)
```

Typzeichen	Bedeutung	Geltungsbereich
%	Integer-Wert	-32768 bis 32767
ohne	Wert mit einfacher Genauigkeit	$10 \cdot 10^{-38}$ bis $10 \cdot 10^{38}$, 7 Stellen hinter dem Komma
!	Wert mit einfacher Genauigkeit	$10 \cdot 10^{-38}$ bis $10 \cdot 10^{38}$, 7 Stellen hinter dem Komma
#	Wert mit doppelter Genauigkeit	$10 \cdot 10^{-38}$ bis $10 \cdot 10^{38}$, 16 Stellen hinter dem Komma
\$	Zeichenvariable (String)	255 Zeichen

Die unter GW-Basic zulässigen Variablentypen

Diese Zeile wandelt die Zahl 1234 aus der Variablen »test%« in eine Zeichenkette um und weist sie der Variablen »zeichen\$« zu.

Mit der Funktion »string\$« kann der Anwender ein Programm dazu veranlassen, selbständig eine Zeichenkette zu generieren:

```
90 x$=string$(80,"*")
```

In dieser Zeile generiert das Programm eine Zeichenkette, die aus 80

Sternchen besteht und weist sie der Variablen »x\$« zu.

In vielen Programmiersprachen gestaltet sich das Vertauschen der Inhalte zweier Variablen aufwendig und umständlich:

```
100 tmp$=test1$ : test1$=test2$ :
test2$=tmp$
```

Sie benötigen eine zusätzliche temporäre Variable »tmp\$« zum Vertauschen der Inhalte von »test1\$« und

»test2\$«. In GW-Basic genügt hierzu der Befehl »swap«:

```
100 swap test1$,test2$
```

In diesem Beispiel werden die Inhalte der Variablen »test1\$« und »test2\$« vertauscht. Die Reihenfolge ist dabei unerheblich.

Im abgedruckten Listing sehen Sie alle beschriebenen Befehle noch einmal mit Kurzkommentaren.

(Bernhard Hecker/tr)

```
[874] 10 'Alle Variablen löschen
[M31] 20 CLEAR
[5N5] 30 'Variable A mit dem Wert 10 belegen
[6W0] 40 A=10
[XK7] 50 'Variablenfeld A mit 20 Plätzen reservieren
[9E1] 60 DIM A(20)
[HU6] 70 'und dem 16. Platz den Wert 1234 zuweisen
[NG1] 80 A(16)=1234
[694] 90 'Eine Test-Zeichenkette
[PH7] 100 TEST$="Das internationale Computermagazin"
[6E6] 110 'Diese in ihre Bestandteile zerlegen
[9P3] 120 TEST1$=LEFT$(TEST$,3)
[YW3] 130 TEST2$=MID$(TEST$,5,14)
[024] 140 TEST3$=RIGHT$(TEST$,15)
[MC7] 150 'Zur Kontrolle auf dem Bildschirm ausgeben
[5F9] 160 PRINT TEST$ : PRINT TEST1$ : PRINT TEST2$ : P
      RINT TEST3$
```

```
[7V7] 170 'Wir suchen nach der Zeichenfolge "Computer"
[RP5] 180 PRINT INSTR(5,TEST$,"Computer")
[1F6] 190 'Wie lang ist die Zeichenkette test$?
[X23] 200 PRINT LEN(TEST$)
[DY6] 210 'Umwandlung eines Strings in eine Zahl
[UL4] 220 AS="1234 TEST" : PRINT VAL(AS)
[TV6] 230 'Umwandlung einer Zahl in einen String
[EU3] 240 ZEICHEN$=STR$(TEST$)
[HG5] 250 'GW-Basic erzeugt 80 Sternchen
[0W2] 260 X$=STRING$(80,"*")
[C27] 270 'Vertauschen zweier Variableninhalte:
[A36] 280 'Einmal auf die umständliche Art
[1X6] 290 TMP$=TEST1$ : TEST1$=TEST2$ : TEST2$=TMP$
[8V5] 300 'und einmal mit dem Swap-Befehl
[TA3] 310 SWAP TEST1$,TEST2$
```

Listing. »vars.bas« zeigt Beispiele zur Variablendefinition unter GW-Basic

GW-Basic-Referenz: Arithmetik

Ohne arithmetische Funktionen kann Ihr Computer weder Rechnen noch mathematische Funktionen bearbeiten. Wie jede andere Programmiersprache kennt auch GW-Basic viele Routinen, die in der Mathematik weiterhelfen.

Arithmetische Funktionen dienen unter anderem mathematischen Berechnungen oder Umwandlungen verschiedener Zahlenformate. Bei Operatoren, die zwischen zwei Ausdrücken stehen (zum Beispiel »and«, »or« und so weiter) müssen Sie ein Leerzeichen vor und nach dem Operator einfügen (»a = b and c«). GW-Basic würde sonst den gesamten Ausdruck als Variablenangabe betrachten (»a = bandc«) – eine häufige Fehlerquelle in selbstgeschriebenen Programmen.

Ein arithmetischer Trick von GW-Basic erlaubt Ihnen, auch innerhalb von Berechnungen Werteabfragen zu verwenden. Ein Beispiel: Der Variablen A soll stets der Wert 20 zugewiesen werden, nur wenn B den Inhalt 10 hat, soll A die Zahl 100 annehmen:

```
a=20+(80*abs(b=10))
```

Die äußeren Klammern sind hier eigentlich nicht nötig, da die Multiplikation sowieso vor der Addition aus-

geführt wird. Sie schaffen aber etwas Übersicht. Wenn B ungleich 10 ist, ergibt der Ausdruck »b = 10« den Wert 0 (logisch unwahr) und hat der Term »abs(b = 10)« das Ergebnis 0. A erhält folglich den Wert 20 + 80x0, also 20.

Sobald B den Inhalt 10 hat, gibt der Ausdruck »b = 10« den Wert -1 (logisch wahr) zurück. Durch die Abs-Funktion ignoriert die Formel das Minuszeichen, und in den äußeren Klammern steht das Ergebnis 80x1. A bekommt also den Wert 20 + 80x1 (100) zugewiesen.

Ein weiteres, häufig auftretendes Problem ist das Hin- und Herschalten eines Variablenwertes zwischen zwei Zahlen. Angenommen, die Variable A soll bei jedem Durchlaufen der Zeile 100 zwischen den Werten 3 und 20 wechseln. Normalerweise benötigt man dazu zwei If-Abfragen und eine Goto-Anweisung:

```
100 if a=3 then a=20 : goto 120
```

```
110 if a=20 then a=3
120 ...
```

Die Goto-Anweisung ist wichtig, weil sonst der Inhalt von A nie von 3 auf 20 schalten würde. Es geht aber auch wesentlich kürzer:

```
100 a=23-a
```

Hat die Variable A beim Erreichen der Zeile 100 den Inhalt 3, dann wird ihr jetzt der Wert 23-3, also 20 zugewiesen. Ist der Inhalt 20, lautet der neue Wert 23-20, also 3.

Viele dieser Programmiertricks finden Sie auch durch das genaue Studium von Listings, die in diesem Sonderheft veröffentlicht sind (»Was macht das Programm in dieser Zeile?«).

Es folgt nun eine Liste der arithmetischen Funktionen von GW-Basic mit Beispielen.

< Ausdruck1 > AND < Ausdruck2 >

Diese Funktion verknüpft die numerischen Werte der beiden Ausdrücke bitweise. An den Stellen, an denen in beiden Ausdrücken ein gesetztes Bit existiert, erhält auch das Ergebnis ein gesetztes Bit:

```
0 and 0 = 0
0 and 1 = 0
1 and 0 = 0
1 and 1 = 1
```

Vor und nach der And-Funktion muß ein Leerzeichen stehen, sonst meldet

GW-Basic einen Syntaxfehler.

Beispiel:

```
if (a and 1)=1 then print "Die Zahl ist un-
gerade"
print 100 and 98
```

ABS(< Ausdruck >)

Ermittelt den absoluten Wert des Ausdrucks, also den Betrag ohne Vorzeichen:

```
abs(-x) = x
abs(0) = 0
abs(+x) = x
```

Der Variablentyp, den die Abs-Funktion zurückliefert, entspricht dem Typ, der auch im Ausdruck verwendet wird.

Beispiel:

```
a=abs(100*wert)
a#=abs(float #)
```

ATN(< Stelle >)

Bestimmt den Wert des Arkustangens an der angegebenen Stelle. Der Arkustangens ist der Kehrwert des Tangens, also mathematisch 1 geteilt durch Tangens. Der ermittelte Wert wird im Bogenmaß zurückgegeben, liegt also zwischen -Pi/2 und Pi/2. Der Funktionswert wird in Form einer einfachgenauen Fließkommazahl zurückgeliefert.

Beispiel:

```
a=atn(1.25)
print atn(pi)
```

CDBL(< Ausdruck >)

Konvertierung des Ausdrucks in eine Fließkommazahl doppelter Genauigkeit.

Beispiel:

```
if float # =cdbl(a%) then flag%=0
print cdbl(1/3)
```

CINT(< Ausdruck >)

Diese Funktion konvertiert einfach- und doppelgenaue Dezimalzahlen in die entsprechenden Ganzzahlen. Liegt der Ausdruck außerhalb des für Ganzzahlen erlaubten Bereichs, so ermittelt die Funktion eine möglichst genaue Ganzzahl.

Beispiel:

```
if cint(a#)=100 then print "100"
print a#cint(a#)
```

CSNG(< Ausdruck >)

Wandelt den Ausdruck in eine Fließkommazahl einfacher Genauigkeit um.

Beispiel:

```
print csng(100.12)
if csng(a%)=1.0E3 then print "ja, es ist
100"
```

COS(< Stelle >)

Diese Funktion ermittelt den Wert des Kosinus an der angegebenen Stelle. Dabei wird die Stelle als Winkel in Bogenmaß angenommen. Zurück-

gegeben wird ein Wert zwischen -1 und 1 als einfachgenaue Fließkommazahl.

Beispiel:

```
print cos(pi)
```

EXP(< Stelle >)

Bestimmt den Wert der Eulerschen Funktion »e« an der angegebenen Stelle. Das Ergebnis wird in Form einer einfachgenauen Fließkommazahl zurückgeliefert.

Beispiel:

```
a=exp(2)
if exp(20)=test! then print "Werte sind
gleich"
```

FIX(< Ausdruck >)

Diese Funktion bestimmt den ganzzahligen Anteil des Ausdrucks. Dabei wird das Ergebnis in dem Variablenformat zurückgegeben, das auch im Ausdruck verwendet wurde. Fix liefert einen um 1 größeren Wert als »int«, wenn negative Argumente verwendet werden.

Beispiel:

```
if fix(a#)=1.0E4 then gosub 1000
gs!=fix(einfach!)
```

INT(< Ausdruck >)

Ermittelt den ganzzahligen Wert des Ausdrucks. Im Gegensatz zur Fix-Funktion, die bei einem positiven Ausdruck das gleiche Ergebnis liefert, benutzt die Int-Funktion die Gauß-Skala. Dadurch ergibt sich bei einem negativen Ausdruck ein um 1 niedriger Wert von »int(< Ausdruck >)< gegenüber »fix(< Ausdruck >)<.

Beispiel:

```
a=int(100.23)
if int(a)=10 then stop
```

LOG(< Ausdruck >)

Bestimmt den natürlichen Logarithmus des Ausdrucks zur Basis »e«. Die Log-Funktion ist die Umkehrfunktion zur Exp-Funktion. Als Ergebnis liefert »log« eine einfachgenaue Fließkommazahl. Soll der Logarithmus zu einer anderen Basis (zum Beispiel 10) berechnet werden, so ist die Formel »log(x)/log(< Basis >)< zu verwenden.

Beispiel:

```
if log(a%)=e then print "Ja, Ende er-
reicht!"
for i%=1 to 10:print log(i%):next
```

< Ausdruck 1 > MOD

< Ausdruck 2 >

Diese Funktion ermittelt den Rest der Division von Ausdruck 1 durch Ausdruck 2.

Beispiel:

```
if (a% mod 9)=8 then print "Rest von 8"
for i%=1 to 10 : print (i% mod 5) : next
```

NOT(< Ausdruck >)

Gibt als Ergebnis das binäre Einerkomplement des Ausdrucks wieder. Dabei werden in der binären Schreibweise alle gesetzten Bits gelöscht und umgekehrt. Weil GW-Basic Zahlen in Vorzeichen-Betrag-Schreibweise speichert, ist »not(a)< gleich »-a-1<.

Beispiel:

```
if not(a)=-8 then print "ja"
for i%=1 to 10 : print not(i%) : next
```

< Ausdruck 1 > OR < Ausdruck 2 >

Verknüpft die binären Argumente »Ausdruck 1« und »Ausdruck 2« bitweise. Im Ergebnis dieser Berechnung werden alle Bits, die in »Ausdruck 1« oder beziehungsweise und in »Ausdruck 2« gesetzt sind, ebenfalls gesetzt:

```
0 or 0 = 0
0 or 1 = 1
1 or 0 = 1
1 or 1 = 1
```

Beispiel:

```
for i%=1 to 20 : print i% or 1 : next
if (za% or 1)=za% then print "Die
Zahl"za%"ist ungerade"
```

SGN(< Ausdruck >)

Ermittelt das Vorzeichen des Ausdrucks. Ergibt die Auswertung einen positiven Wert, so liefert die Sgn-Funktion den Wert 1, ergibt sie einen negativen Wert, liefert sie -1, ansonsten 0:

```
sgn(-x) = -1
sgn(0) = 0
sgn(+x) = 1
```

Beispiel:

```
print sgn(-1200)
if sgn(val(left$(f1$,1)))=-1 then 2000
```

SIN(< Stelle >)

Bestimmt den Sinus an der angegebenen Stelle. Diese wird als Wert im Bogenmaß erwartet. Die Sin-Funktion liefert als Ergebnis einen Wert zwischen -1 und 1 zurück.

Beispiel:

```
if sin(winkel%)=1 then print "Der Winkel
ist 90 oder 180 Grad groß"
sinus%=sin(arg%)
```

SQR(< Ausdruck >)

Ermittelt die Quadratwurzel des Ausdrucks. Das Ergebnis ist eine Fließkommazahl von einfacher Genauigkeit.

Beispiel:

```
if sqr(za) < 4 then print "Die
Zahl"za;"ist kleiner als 16"
sta=sqr(ta)
```

TAN(< Stelle >)

Liefert den Wert des Tangens an der angegebenen Stelle. Das Funktionsargument wird als Wert in Bogenmaß angegeben und das Ergebnis als ein-

fachgenaue Fließkommazahl zurückgeliefert.

Beispiel:

```
if tan(winkel)=pi then print "Der Winkel
ist 45 Grad groß"
tza=tan(za)
```

VAL(< Ausdruck >)

Wandelt den Zeichenkettenausdruck Ausdruck in einen numerischen Wert um. Dabei werden alle Zeichen von Beginn der Zeichenkette Ausdruck an umgewandelt. Beim ersten Zeichen, das sich nicht in eine Zahl umwandeln läßt, wird die Konvertierung abgebrochen.

Beispiel:

```
if val(za$) < 0 then gosub 1000
wert=100*val(wert$)
print val("100")
```

< Ausdruck1 > XOR < Ausdruck2 >
Verknüpft die beiden Ausdrücke bitweise durch die logische Xor-Funktion (logisches »exklusiv oder«).

»XOR« bedeutet im Gegensatz zur Or-Funktion »oder« und schließt »und« aus:

```
0 xor 0 = 0
0 xor 1 = 1
1 xor 0 = 1
1 xor 1 = 0
```

Beispiel:

```
print 255 xor 128
a=(b xor 15)
```

(Martin Althaus/tr)

Tip: Umrechnung von Winkeln zwischen dem Grad- und Bogenmaß

Die Winkelfunktionen »sin«, »cos«, »tan« und »atn« arbeiten mit Werten im Bogenmaß. Ein mathematischer Vollkreis durchläuft die Werte von 0 bis 2 Pi. Die Umrechnung ins bei uns übliche Gradmaß, in dem ein Vollkreis zwischen 0 und 360 Grad liegt, geht so:

```
Winkel im Gradmaß=Winkel im Bogenmaß x  $\frac{360}{2 \pi}$ 
Winkel im Bogenmaß=Winkel im Gradmaß x  $\frac{2 \pi}{360}$ 
```

Da GW-Basic leider keine Konstante Pi fest gespeichert hat, sollte man sich am Programmmanfang eine Variable bereitlegen:

```
10 Pi# = 3.141592653589793
```

Und die Umrechnung läßt sich als Basic-Funktion programmieren:

```
20 def fn grad(x)=x*180/pi#
30 def fn bogen(x)=x*pi#/180
```

Beispiel: 40 print "100 Grad sind im Bogenmaß:";fn bogen(100)

GW-Basic-Referenz: Event-Handling

Event heißt auf deutsch »Ereignis«, und »handling« läßt sich mit »Handhabung, Steuerung« übersetzen. Auf den Computer bezogen, sind Events Unterprogrammaufrufe, die erst bei Eintreten eines bestimmten Ereignisses ausgeführt werden. Dies kann zum Beispiel das Erreichen einer bestimmten Uhrzeit sein oder auch das Drücken eines Feuerknopfs am Joystick.

»Events« sind Programmaufrufe, die unabhängig von der bearbeiteten Programmstelle, durch andere (äußere) Ereignisse erfolgen. Die Besonderheit dabei liegt darin, daß das Programm nicht ständig entsprechende If-Abfragen durchlaufen muß. Vielmehr unterbricht der GW-Basic-Interpreter das laufende Programm, gleichgültig, was es gerade ausführt, verzweigt zu einer vorher definierten Unterroutine und setzt danach den Programmablauf an der Unterbrechungsstelle fort. Das hört sich komplizierter an, als es in Wirklichkeit ist.

Der einfachste Event ist der Error-Event. Mit der Zeile

```
on error gosub <Zeile>
```

legen Sie in Ihrem Programm fest, zu welcher »Zeile« GW-Basic verzweigen soll, wenn ein Programmfehler auftritt. Normalerweise bricht der Interpreter bei einem Fehler das Programm ab, was durch den On-Error-Event verhindert wird. In der Unteroutine ab »Zeile« läßt sich dann eine entsprechende Fehlerbehandlung programmieren, die je nach Fehlerursache (steht in der Systemva-

riablen »err«) zum Beispiel einen Hinweistext ausgibt.

ON COM(< Schnittstellennummer >) GOSUB < Zeilennummer >

Liegt am seriellen Port mit der angegebenen Schnittstellennummer ein Zeichen zur Verarbeitung an, so löst GW-Basic einen On-Com-Event aus und verzweigt zur Unteroutine ab der Zeilennummer.

Nach Beendigung des Unterprogramms mit einer Return-Anweisung wird das Programm an der Stelle fortgesetzt, an der die Unterbrechung aufgetreten ist. Dadurch lassen sich die Schnittstellen abfragen, um zum Beispiel ein Programm zur Datenfernübertragung zu schreiben.

```
10 on com (1) gosub 500
20 on com (a) gosub 10000
```

COM(< Schnittstellennummer >) ON | OFF | STOP

Um die periodische Abfrage der seriellen Schnittstelle durch das On-Com-Kommando in ein Programm einzubinden, legen Sie die Nummer der seriellen Schnittstelle fest, für die die Abfrage durchgeführt werden soll. Der zweite Parameter teilt GW-

Basic mit, ob die Abfrage eingeschaltet (»on«), ausgeschaltet (»off«) oder unterbrochen (»stop«) werden soll. In letzterem Fall speichert der Interpreter Unterbrechungsanforderungen bis zum nächsten »com(...) on«.

```
10 com (1) off
20 com (a) on
```

ON ERROR GOTO < Zeilennummer >

Sie können sich in Basic eine eigene Routine zur Fehlerbehandlung schreiben. Dadurch wird verhindert, daß bei Auftreten eines Fehlers das Programm mit einer Fehlermeldung abgebrochen wird. Der Parameter »Zeilennummer« gibt an, an welche Zeile der Interpreter die Kontrolle übergeben soll, wenn im Verlauf des Programms ein Fehler auftritt. Eine Routine für die Fehlerbehandlung schließen Sie mit dem Resume-Befehl ab. Mit dem Kommando

```
on error goto 0
```

wird die Routine zur Fehlerbehandlung wieder ausgeschaltet.

```
10 on error goto 1000
20 on error goto 500
```

ERL

Ein Unterprogramm zur Fehlerbehandlung, das mit dem On-Error-Goto-Befehl festgelegt wurde, kann durch die Systemvariable »erl« die Nummer der Programmzeile ermitteln, in der der Fehler auftrat.

```
100 a=erl
110 print "Der Fehler trat in Zeile "a" auf."
```

ERR

In der Systemvariablen »err« findet eine On-Error-Unteroutine die Nummer des zuletzt aufgetretenen Feh-

lers. In der Tabelle 1 finden Sie alle Fehlermeldungen von GW-Basic mit den entsprechenden Fehlernummern für die Err-Variable.

```
10 fehlerflag(err)=1
20 print "Es ist Fehler Nr."err"im Programm aufgetreten."
```

RESUME [<Zeilennummer>**] **[NEXT] [0]****

Am Ende einer Routine zu Fehlerbehandlung, die mit dem On-Error-Goto-Befehl gesetzt wurde, muß ein Resume-Kommando folgen, um die Bearbeitung des laufenden Programms wieder aufzunehmen. Der Parameter »Zeilennummer« gibt an, von welcher Programmzeile an GW-Basic die Bearbeitung weiterführen soll. Wird statt dessen das Schlüsselwort »next« hinter den Resume-Befehl gesetzt, so wird die Bearbeitung an dem Befehl fortgesetzt, der direkt der fehlerverursachenden Anweisung folgt. Mit dem Wert Null als Zeilennummer wird der Befehl wiederholt, der den Fehler verursacht hat.

```
10 resume 100
20 resume next
```

ON KEY(<Tastennummer>**) GOSUB **<Zeilennummer>****

Bestimmte Tasten der PC-Tastatur können unter Basic gesondert abgefragt werden. Immer, wenn Sie in einem laufenden Programm die Taste mit dem Tasten-Code »Tastennummer« drücken, verzweigt das Programm in ein Unterprogramm ab Zeile »Zeilennummer«. Dieses Unterprogramm muß durch ein Return-Kommando beendet werden. Nach der Ausführung des Unterprogramms setzt der Interpreter die Ausführung an der Stelle fort, an der das Programm durch den Tastendruck unterbrochen wurde, wodurch Sie also Tasten »im Hintergrund« abfragen können. Die zulässigen Codes für die Tastennummer finden Sie in Tabelle 2.

```
10 on key (1) gosub 100
20 key (1) on
30 print "Bitte F1 drücken!"
40 goto 30
100 print "F1 gedrückt!"
110 end
```

KEY(<Tastennummer>**) ON| OFF **STOP****

Zum Einbinden einer periodischen Abfrage der Tastatur durch das On-Key-Kommando in ein Programm legen Sie die Tastennummer fest, für die die Abfrage durchgeführt werden soll (Tabelle 2). Mit dem zweiten Parameter geben Sie an, ob die Abfrage für die angegebene Taste eingeschaltet (»on«), ausgeschaltet (»off«) oder

unterbrochen (»stop«) werden soll. In letzterem Fall speichert der Interpreter Unterbrechungsanforderungen bis zum nächsten »key(x) on«.

```
10 key (10) ON
```

KEY **<Code> ,CHR\$ **(**<Status>**) + CHR\$ **(**<Scancode>**)******

Definition für die reservierten Tastaturcodes 15 bis 20 aus Tabelle 2. Für den Status setzen Sie eine der Hexadezimalzahlen aus Tabelle 3 ein, wobei auch Kombinationen daraus erlaubt sind. Der Scancode ist der Dezimalwert der jeweiligen Taste (ohne die Zusatztasten). Um zum Beispiel [Alt-F1] abzufragen, verwenden Sie die Anweisung

key 15,chr\$(&h08)+chr\$(59)
59 ist dabei der Scancode der Taste [F1]. Ein Beispielprogramm zum Abschalten der Tastenkombinationen [Alt-Ctrl-Del] und [Ctrl-Break] finden Sie in Listing 1.

ON PEN GOSUB **<Zeilennummer>**
Bei einem angeschlossenen Schreib-/Lesestift (Lightpen) kann Basic überprüfen, ob der Anwender die Taste am Stift gedrückt hat. Dazu legen Sie über »Zeilennummer« den Start eines Unterprogramms fest, mit dem der Interpreter in einem solchen Fall die Programmausführung fortsetzen soll. Dieses Unterprogramm muß mit einem Return-Befehl enden.

```
10 on pen gosub 1000
```

1	NEXT without FOR NEXT ohne FOR	29	WHILE without WEND WHILE ohne WEND
2	Syntax Error Syntaxfehler	30	WEND without WHILE WEND ohne WHILE
3	RETURN without GOSUB RETURN ohne GOSUB	50	Field Overflow FIELD-Überlauf
4	Out of data Zu wenige DATA-Anweisungen	51	Internal Error Interner Fehler
5	Illegal function call Unzulässiger Funktionsaufruf	52	Bad File Number Ungültige Dateinummer
6	Overflow Überlauf	53	File Not Found Datei nicht gefunden
7	Out of Memory Speicher voll	54	Bad File Mode Ungültige Dateioption
8	Undefined Line Number Nichtdefinierte Zeilennummer	55	File Already Open Datei bereits geöffnet
9	Subscript Out of Range Indexbereich nicht eingehalten	57	Device I/O Error Ein-/Ausgabefehler
10	Duplicate Definition Doppelte Definition	58	File Already Exists Datei bereits vorhanden
11	Division by Zero Division durch Null	61	Disk Full Diskette voll
12	Illegal Direct Unzulässige Direkt-Anweisung	62	Input past End Einlesen nach Dateende
13	Type Mismatch Unverträgliche Variablentypen	63	Bad Record Number Unzulässige Dateisatz-Nummer
14	Out of String Space Zeichenfolgen-Speicher voll	64	Bad File Name Unzulässiger Dateiname
15	String Too Long Zeichenfolge zu lang	66	Direct Statement in File Direkte Anweisung in Datei
16	String Formula Too Complex Zeichenfolgen-Formel zu komplex	67	Too Many Files Zu viele Dateien
17	Can't Continue Fortsetzung nicht möglich	68	Device Unavailable Gerät nicht verfügbar
18	Undefined User Function Nichtdefinierte Funktion	69	Communication Buffer Overflow Kommunikationspuffer-Überlauf
19	No RESUME Kein RESUME	70	Disk Write Protect Zugriff verweigert
20	RESUME without Error RESUME ohne Fehler	71	Disk Not Ready Laufwerk nicht bereit
22	Missing Operand Fehlender Operand	72	Disk Media Error Speichermedium-Fehler
23	Line Buffer Overflow Zeile zu lang	73	Advanced Feature Reserviertes Wort
24	Device Timeout Zeitlimit-Überschreitung	74	Rename across Disks NAME auf mehrere Disketten angewendet
25	Device Fault Gerätefehler	75	Path/File Access Error Fehler bei Datei-/Suchwegzugriff
26	FOR without NEXT FOR ohne NEXT	76	Path Not Found Suchweg nicht gefunden
27	Out of Paper Papierende		

Tabelle 1. Die englischen und deutschen Fehlermeldungen von GW-Basic mit der entsprechenden Fehlernummer

PEN ON | OFF | STOP

Steuert die On-Pen-Anweisung. Die Abfrage des Lichtgriffels läßt sich einschalten (»on«), ausschalten (»off«) oder unterbrechen (»stop«). Dazu rufen Sie den Pen-Befehl mit einem dieser drei Parameter auf. In letzterem Fall speichert der Interpreter Unterbrechungsanforderungen bis zum nächsten »pen on«.

10 pen on

ON PLAY(<Notenanzahl>) GOSUB <Zeilennummer>

Abhängig von der Anzahl der Noten, die sich noch im internen Musikpuffer des Interpreters befinden, verzweigt GW-Basic mit diesem Kommando zu einem Unterprogramm ab der angegebenen Zeilennummer, wenn noch genau »Notenanzahl« (1 bis 32) Noten zu spielen sind. Die Abfrage muß mit dem Play-On-Befehl in ein Programm eingebunden werden. Eine kleine Demonstration dieser Fähigkeiten zeigt Listing 2.

Code	Taste	Code	Taste
1	F1	9	F9
2	F2	10	F10
3	F3	11	[Cursor oben]
4	F4	12	[Cursor links]
5	F5	13	[Cursor rechts]
6	F6	14	[Cursor unten]
7	F7	15 bis	reserviert (Tabelle 3)
8	F8	20	

Tabelle 2. Die Funktionstastencodes für den On-Key-Befehl

PLAY ON | OFF | STOP

Die Abfrage der Anzahl der noch im Musikpuffer befindlichen Noten des Interpreters kann mit diesem Kommando eingeschaltet (»on«), ausgeschaltet (»off«) oder unterbrochen (»stop«) werden. In letzterem Fall speichert der Interpreter Unterbrechungsanforderungen bis zum nächsten »play on«.

ON STRIG(<Feuerknopf>)

GOSUB <Zeilennummer>
Event zur kontinuierlichen Abfrage eines Joysticks.

Der Parameter »Zeilennummer« bestimmt die Startzeile des betreffenden Unterprogramms, in das GW-Basic verzweigt, wenn der Anwender den mit dem Parameter »Feuerknopf« gewählten Joystick-Knopf betätigt.

Ein Wert von 0 oder 2 steht für die Knöpfe an Joystick 1, ein Wert von 4 oder 6 für die Knöpfe an Joystick 2.

10 on strig(2) gosub 10000

&H40 [Caps Lock] eingeschaltet
&H20 [Num Lock] eingeschaltet
&H08 [Alt] gedrückt
&H04 [Ctrl] gedrückt
&H02 [Shift links] gedrückt
&H01 [Shift rechts] gedrückt
&H00 [Caps Lock] und [Num Lock] ausgeschaltet
Kombinationen aus diesen Codes sind zulässig.

Tabelle 3. Zustandscodes für erweiterte On-Key-Abfragen

STRIG ON | OFF | STOP

Dient dem Steuern eines mit dem On-Strig-Kommando gesetzten Events. Die Abfrage läßt sich einschalten (»on«), ausschalten (»off«) oder unterbrechen (»stop«). In letzterem Fall speichert der Interpreter Unterbrechungsanforderungen bis zum nächsten »strig on«.

10 strig off

ON TIMER (<Sekundenanzahl>) GOSUB <Zeilennummer>

Basic erlaubt es, Unterprogramme periodisch aufrufen zu lassen. Dazu wird mit dem Parameter »Sekundenanzahl« die Zeit angegeben, die zwischen zwei Aufrufen dieses Unterprogramms verstreichen muß. Alle »Sekundenanzahl« Sekunden wird dann das angegebene Unterprogramm aufgerufen. Der Wert für die Sekunden darf sich im Bereich von 1 und 86400 (24 Stunden) bewegen.

10 on timer(20) gosub 1200
20 on timer(zeit%) gosub 20000

TIMER ON | OFF | STOP

Sie können die Ausführung eines mit dem On-Timer-Befehl definierten, periodischen Unterprogramms mit diesem Kommando einschalten (»on«), ausschalten (»off«) oder unterbrechen (»stop«). In letzterem Fall speichert der Interpreter Unterbrechungsanforderungen bis zum nächsten »timer on«.

10 timer on

(Martin Althaus/tr)

```
[D53] 10 ON KEY (15) GOSUB 100
[M53] 20 ON KEY (16) GOSUB 200
[V64] 30 KEY 15,CHR$(&H8+&H4)+CHR$(83)
[DU1] 40 KEY (15) ON
[BP3] 50 KEY 16,CHR$(&H4)+CHR$(70)
[MU1] 60 KEY (16) ON
[LR5] 70 PRINT "Mich unterbricht niemand!!!"
[XD1] 80 GOTO 80
[3P6] 100 PRINT"[Alt-Ctrl-Del] nützt nichts!"
[VL1] 110 RETURN
[SC6] 200 PRINT"[Ctrl-Break] nützt nichts!"
[XL1] 210 RETURN
```

Listing 1. [Alt-Ctrl-Del] und [Ctrl-Break] abschalten mit »abschalt.bas«

```
[CP9] 10 P$="CDEFGABAGFED":PI%=4:PL%=LEN(P$):PLAY "MB O
2 T200 "+"LEFT$(P$,3)
[3D6] 20 ON PLAY(3) GOSUB 180:PLAY ON:CLS:ZE$=""
[2T3] 30 O%=2:ON KEY(1) GOSUB 210
[D03] 40 ON KEY(2) GOSUB 240
[R03] 50 ON KEY(3) GOSUB 260
[313] 60 ON KEY(4) GOSUB 290
[RN4] 70 FOR I%=1 TO 4:KEY(I%) ON:NEXT
[VA6] 80 SK%=1:ON TIMER(SK%) GOSUB 310:TIMER ON
[U07] 90 LOCATE 5,1:PRINT "F1 : Oktave erhöhen ( 2 )"
[8D5] 100 PRINT "F2 : Oktave erniedrigen"
[UA9] 110 PRINT "F3 : Pause zwischen zwei Sternchen erhöhen ( 1 )"

```

```
[C89] 120 PRINT "F4 : Pause zwischen zwei Sternchen erniedrigen"
[G4G] 130 LOCATE 10,1:PRINT "Das gesamte Programm läuft im Hintergrund. Bis auf die nächste Zeile, in der"
[42E] 140 PRINT "die Zahl 1 hochgezählt wird, arbeitet das ganze Programm mit On-Befehlen,"
[EKE] 150 PRINT "einschließlich der Tastaturabfrage, der Musik und dem Setzen der Sternchen."
[SE8] 160 LOCATE 11,9:PRINT II%:II%=II%+1:IF II%=32000 THEN II%=1
[WP1] 170 GOTO 160
[XE3] 180 PLAY(MID$(P$,PI%,3))
[P85] 190 PI%=PI%+3:IF PI%>PL% THEN PI%=1
[UL1] 200 RETURN
[T14] 210 O%=O%+1:IF O%>6 THEN O%=6
[3F8] 220 PLAY "O"+RIGHT$(STR$(O%),1)+" ":LOCATE 5,22:PRINT O%;"")
[3M1] 230 RETURN
[GU3] 240 O%=O%-1:IF O%<0 THEN O%=0
[BP1] 250 GOTO 220
[G85] 260 SK%=SK%+1:IF SK%>100 THEN SK%=100
[6W7] 270 LOCATE 7,45:PRINT SK%;"") ":ON TIMER(SK%) GOSUB 310
[KM1] 280 RETURN
[Z05] 290 SK%=SK%-1:IF SK%<=1 THEN SK%=1
[KP1] 300 GOTO 270
[CE2] 310 LOCATE 1,1+SP%
[3H6] 320 SP%=SP%+1:IF SP%=80 THEN SP%=0:GOTO 340
[RC3] 330 PRINT ZE$;:RETURN
[PA5] 340 IF ZE$="" THEN ZE$="" ":ELSE ZE$=""
[BM1] 350 RETURN
```

Listing 2. »musik.bas« demonstriert den On-Play-Befehl

Grafikprogrammierung mit GW-Basic

Ansprechende Grafiken erregen immer und überall die Aufmerksamkeit, nicht nur von Computerbegeisterten. Dies ist wohl darauf zurückzuführen, daß über die Programmierung von Grafiken nur sehr diffuse Vorstellungen bestehen. Mit dem leistungsfähigen Befehls-schatz von GW-Basic ist Grafikprogrammierung jedoch kein Geheimnis.

GW-Basic zeichnet sich dadurch aus, daß sich mit seiner Hilfe ohne großen Aufwand grafische Anwendungen programmieren lassen. Anwenderfreundliche Pull-down-Menüs kennen Sie bereits von diversen Benutzeroberflächen wie zum Beispiel »GEM« oder »Windows«. Solche komfortablen Auswahlmenüs sind jedoch nicht nur Anwendern vorbehalten, die eine solche Benutzeroberfläche besitzen. Auch mit GW-Basic lassen sich diese Pull-down-Menüs realisieren.

GW-Basic unterstützt die verschiedene Grafikmodi der Personalcomputer. Der erste Grafikmodus läßt sich allerdings nur bedingt als Grafikmodus bezeichnen, weil er gleichzeitig der Textdarstellung dient. Der Titel »Grafikmodus« ist nur durch die Tatsache begründet, daß im Textmodus durch Verwendung der IBM-Grafikzeichen sogenannte Blockgrafiken darstellbar sind. GW-Basic kennt in diesem Modus zwei Auflösungsstufen. In Abhängigkeit vom Befehl »width« werden 40 oder 80 Textzeichen pro Bildschirmzeile dargestellt. Sind Sie Besitzer eines Farbmonitors, so stehen Ihnen im Textmodus 16 Farben zur Wahl.

Verfügt Ihr PC über eine CGA- oder EGA-Karte, so lassen sich zwei weitere Grafikmodi benutzen

Verfügt Ihr PC über eine CGA- oder EGA-Grafikkarte, so können Sie zwei weitere Grafikmodi benutzen. Im mittleren Grafikmodus stehen Ihnen dabei 64000 Bildpunkte zur Verfügung. Dabei lassen sich in der Horizontalen 320 und in der Vertikalen 200 Bildpunkte ansprechen. Die Ausrichtung der Bildkoordinaten unterscheidet sich etwas von der aus der Mathematik bekannten Darstellung: Auf dem Grafikbildschirm verlaufen die x-Werte von links nach rechts und die y-Werte von oben nach unten, jeweils in positiver Richtung. Die Zählung beider Koordinaten beginnt bei Null. Bei Verwendung eines Farbmonitors lassen sich in der

mittleren Grafikauflösung maximal vier Farben benutzen.

Im dritten und letzten Modus stehen nur noch zwei Farben zur Wahl. Dafür wird die Grafikauflösung verdoppelt. Bei einer Darstellung von 640 mal 200 Bildpunkten können Sie nun 128000 Bildpunkte einzeln ansprechen. Auch hier beginnt die Zählweise links oben bei Bildpunkt (0,0) und endet unten rechts bei Bildpunkt (639,199).

Das Koordinatensystem von GW-Basic weicht von dem aus der Mathematik bekannten Koordinatensystem ab

In dem Programm aus Listing 1 werden ausschließlich der Textmodus von GW-Basic und die zur Auswahl stehenden IBM-Grafikzeichen verwendet. Es werden Pull-down-Menüs realisiert, und Sie lernen den Grafikbefehlssatz von GW-Basic kennen.

Mit Hilfe von Menüs lassen sich bekanntlich sehr komfortable Auswahlsteuerungen verwirklichen. Ein Pull-down-Menü besteht aus einer Kopfzeile, unter deren Begriffen die einzelnen Menü-Punkte verborgen liegen. Durch Auswahl eines Begriffes wird das entsprechende Menü »ausgerollt«. Nun wird mit Hilfe der Cursor-Tasten die gewünschte Routine aufgerufen.

Die Menüs werden durch die Cursor-Tasten [Cursor links] und [Cursor rechts] ausgewählt. Mit den Tasten [Cursor oben] und [Cursor unten] läßt sich ein Menü-Punkt selektieren. Dieser wird dann invertiert am Bildschirm dargestellt. Die Auswahl selbst erfolgt durch Drücken der Enter-Taste.

Besitzer einer Maus können auch diese als Steuergerät nutzen. Um das Beispielprogramm mit einer sinnvollen Anwendung zu versehen, wurde es als »Grafik-Hilfsprogramm« entwickelt. Wird ein Menü-Punkt (Grafikbefehl) angewählt, so gibt das Programm eine Kurzerklärung dieser Grafikanweisung auf dem Bildschirm aus.

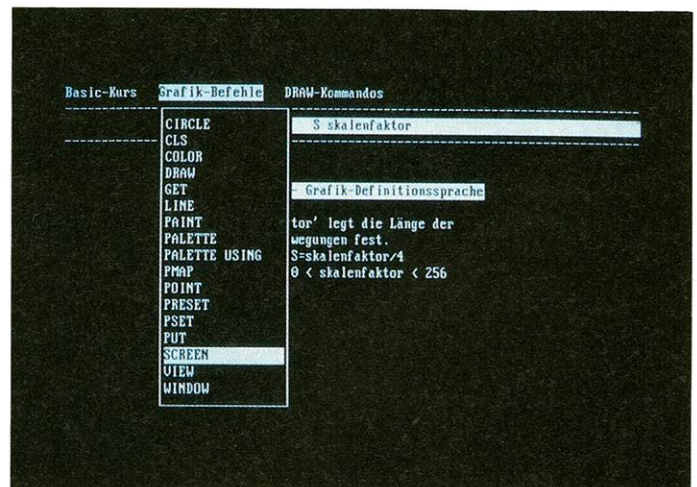
Natürlich soll der Bildschirmhintergrund nach dem »Zuklappen« eines Pull-down-Menüs wieder genauso aussehen wie vorher. Um dies zu erreichen, wird der Bildschirmhintergrund zunächst gespeichert und später wieder (in den Bildschirmspeicher) zurückgeschrieben. Diesem Zweck dient die Feldvariable »hintergrund\$«. Sie wird in den Zeilen 150 bis 170 auf 80 Zeichen pro Feld definiert. In den Feldern 1 bis 25 wird später der gesamte Bildschirm abgelegt.

Bei der Programmierung von Pull-down-Menüs sind einige Voraussetzungen zu beachten

Die eigentliche Routine zur Rettung der Hintergrundinformation steht in den Zeilen 660 bis 730. Die Routine ist nur in der Lage, rechteckige Bildschirmbereiche zu speichern. Mehr ist aber auch nicht erforderlich, da die Pull-down-Menüs ebenfalls rechteckig sind.

Das Unterprogramm benötigt zum Speichern der Bildinformationen die Position der linken oberen Ecke sowie die Länge und Breite des Bild-

Bild 1. Pull-down-Menüs lassen sich auch mit GW-Basic verwirklichen



schirmbereichs. Diese Informationen werden in den Variablen »fenster-spalte«, »startspalte«, »fensterlänge« und »fensterbreite« übergeben. Die Routine liest dann die Bildschirminformation mit Hilfe der Screen-Funktion in die Hintergrund-Variablen ein. Diese nützliche Funktion zum Einlesen von ASCII-Werten einer Bildschirmposition hat folgende Syntax:

```
v = screen(zeile,spalte,<flag>)
```

Die Zählweise im Textmodus beginnt ebenfalls in der linken oberen Ecke (Position 1,1) und endet rechts unten bei Position (25,80) oder (25,40) (im 40-Zeichen-Modus). Wird bei der Screen-Funktion das Flag gesetzt (Wert ungleich Null), so wird anstelle des ASCII-Wertes der Farbwert des Zeichens (Attribut-Byte) gelesen. Da die Routine jede Bildschirmposition einzeln lesen muß (siehe Zeile 700), versteht es sich von selbst, daß das Sichern der Hintergrundinformation etwas Zeit beansprucht. (Für fortgeschrittene Programmierer empfiehlt sich die Verwendung einer Assembler- oder C-Routine zur Beschleunigung des Programmablaufs.) Ist das Unterprogramm abgeschlossen, so befindet sich das Abbild des Bildschirms in den Feldvariablen »hintergrund(1)« bis »hintergrund(25)«.

Die Routine zum Zurückschreiben der Hintergrundinformation ist etwas einfacher (Zeile 750 bis 800). Weil nicht jedes Zeichen einzeln, sondern ganze Zeilen auf einmal zurückgeschrieben werden, arbeitet sie deutlich schneller. Natürlich sind auch hier dieselben Variablen wie bei der Einlese-Routine erforderlich. Mit »locate« wird der Cursor der Reihe nach in jeder Zeile an die Startspalte des zu regenerierenden Bildschirmbereiches gesetzt. Die Anweisung »print...mid\$...« schreibt anschließend den geretteten Bildschirmbereich Zeile für Zeile an seinen ursprünglichen Platz zurück. Die beiden Routinen bleiben nicht nur der Anwendung »Pull-down-Menü« vorbehalten. Sie können dieses Unterprogramm auch dort einsetzen, wo Sie mit Fenstern arbeiten wollen. Bei Verwendung der oben genannten Variablen lassen sich die

Routinen in jedem anderen Programm einsetzen.

Die Menü-Definition steht in den Zeilen 1900 bis 2320. Die Kopfzeile des Menüs wird in der Variablen »kopf\$« festgehalten. Zur Unterscheidung der Menüs müssen diese in der Kopfzeile durch mindestens ein Leerzeichen voneinander getrennt sein. In der Feldvariablen »menue\$()« werden die einzelnen Menü-Punkte gespeichert. Ein Leer-String trennt die verschiedenen Menüleisten voneinander. Alle Einträge eines Pull-down-Menüs müssen dieselbe Länge besitzen. Die Auswertung der Menü-Parameter erfolgt in den Programmzeilen 1660 bis 1880. Im ersten Abschnitt (Zeile 1660 bis 1780) wird die Menü-Anzahl, maximale Menü-Breite und deren Startspalte anhand der Kopfzeile (»kopf\$«) ermittelt. Die Länge der einzelnen Menüs und die Zuordnung der Einträge erfolgt im zweiten Programmabschnitt.

Etwas Verwaltungsarbeit ist beim Umgang mit Pull-down-Menüs schon erforderlich

Zur Verwaltung der Pull-down-Menüs gehört natürlich ein Unterprogramm, das die Menüs am Bildschirm darstellt. Diese Routine bedient sich der Blockgrafik, mit deren Hilfe die Menüs ein professionelles Aussehen erhalten (Bild 1). In den Zeilen 820 bis 920 wird ein Menü ausgerollt und mit einem Rahmen (aus IBM-Grafikzeichen) versehen.

Die Auswahl eines neuen Befehls erfolgt im Unterprogramm »Option auswählen« (Zeile 940 bis 1030). Der selektierte Befehl wird durch die Anweisung »color 0,7« invertiert dargestellt. Der zuletzt angewählte Menü-Punkt wird durch erneutes Überschreiben normalisiert. Schwieriger wird die Behandlung eines neu-

en Pull-down-Menüs. Die entsprechende Routine ab Zeile 1050 stellt zunächst den Original-Hintergrund des letzten Menüs wieder her. Anschließend berechnet die Routine die nötigen Parameter für das neue aktive Menü und gibt dieses am Bildschirm aus. Zuletzt wird die Menü-Überschrift des betreffenden Menüs ebenfalls invertiert dargestellt.

Natürlich muß das Programm auch Benutzereingaben erkennen und entsprechend behandeln. Dieses Unterprogramm steht in den Zeilen 1410 bis 1550. Die While-Wend-Schleife wird so lange durchlaufen, bis Sie die Enter-Taste drücken. Die Cursor-Tasten veranlassen das Programm, die Menüs oder die Befehle zu wechseln. Erst nach Betätigung der Enter-Taste wird im Programm fortgefahren.

Die Routine übergibt dem Hauptprogramm die Parameter »menue« und »befehl«. »menue« enthält das aktuelle Pull-down-Menü, während »befehl« den ausgewählten Menü-Punkt übermittelt. Anhand dieser beiden Parameter bestimmt das Hauptprogramm die zugehörigen Unterprogramme und führt sie aus (Zeile 420 bis 630). Um das Programm nicht unnötig kompliziert zu gestalten, wurden die Unterprogramme als reine Kommentar-Anweisungen programmiert. Wollen Sie sich also eine Kurzerklärung eines Grafikbefehls anzeigen lassen, so genügt es, den betreffenden Befehl mit den Cursor-Tasten oder der Maus anzuwählen und anschließend mit der Maus- oder Enter-Taste zu bestätigen.

Die einzelnen Routinen des Programms sind so ausgelegt, daß Sie diese in Ihre eigenen Programme einbinden können. Die Anzahl der Menüs und deren Menü-Punkte hängt nicht vom Programm, sondern lediglich von der Definition der Variablen

	GW-Basic	HBasic
Auflösung	640x200 Punkte	720x348 Punkte
Bildpunkte insgesamt	128000 Bildpunkte	250560 Bildpunkte
Monitor-Typ	Farbmonitor	Monochrom-Monitor
Textdarstellung	80x25 Zeichen	80x25 Zeichen
Zeichenmatrix	8x8 Punkte	9x14 Punkte

Tabelle 1. Unterscheidungsmerkmale von HBasic und GW-Basic

Farbnummer	Farbmonitor	Monochrom-Monitor
0	Schwarz	Schwarz
1	Blau	Weiß und unterstrichen
2	Grün	Weiß
3	Kobaltblau	Weiß
4	Rot	Weiß
5	Violett	Weiß
6	Braun	Weiß
7	Weiß	Weiß
8	Grau	Schwarz
9	Hellblau	Intensiv-Weiß und unterstrichen
10	Hellgrün	Intensiv-Weiß
11	Hellkobaltblau	Intensiv-Weiß
12	Hellrot	Intensiv-Weiß
13	Hellviolett	Intensiv-Weiß
14	Gelb	Intensiv-Weiß
15	Intensiv-Weiß	Intensiv-Weiß

Tabelle 2. Die Farbnummern bestimmen die Farbe oder Helligkeit auf dem Bildschirm

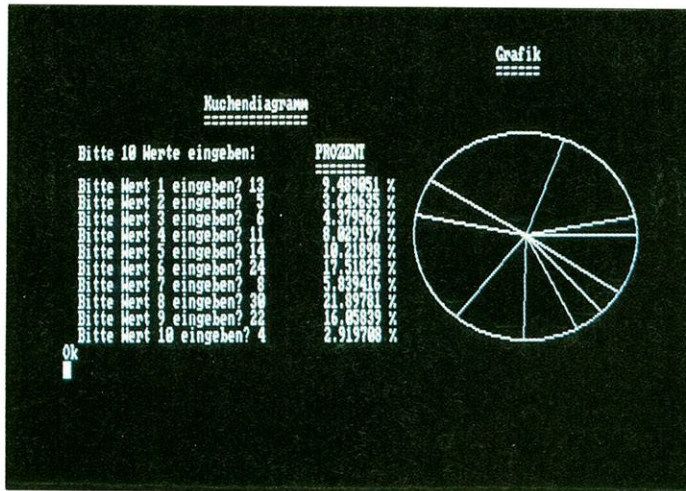


Bild 2. Ein Kuchendiagramm läßt sich bereits mit wenigen Anweisungen erzeugen



Bild 3. Mit dem Befehl »draw« lassen sich beachtliche Dinge realisieren

»kopf\$« und »menue\$()« ab. Es müssen dabei nur folgenden Bedingungen erfüllt sein:

- Die Menü-Überschriften der Kopfzeile müssen durch mindestens ein Leerzeichen voneinander getrennt sein (Zeile 1920).
- Die Variable »kopf\$« muß mit einem Leerzeichen enden (Zeile 1920).
- Die einzelnen Menü-Punkte müssen durch einen Leer-String voneinander getrennt sein (Zeilen 1930, 1970, 2150, 2310).
- Alle Menü-Punkte eines Pull-down-Menüs müssen gleich lang sein. Ist dies nicht der Fall, so sind alle Einträge mit Leerzeichen auf dieselbe Länge zu bringen.

Basic ist keineswegs nur für Einsteiger gedacht – auch komplizierte Aufgaben sind realisierbar

Die beiden Routinen zum Retten und Zurückschreiben des Bildschirmhintergrundes lassen sich ebenso in Programmen mit Fensterverwaltung nutzen. Es ist lediglich zu beachten, daß das Unterprogramm zum Retten des Bildschirms wegen der vielen einzelnen Bildschirmabfragen sehr langsam abläuft. Abhilfe schafft in diesem Fall ein Basic-Compiler, der den GW-Basic-Code direkt in Maschinensprache übersetzt.

Spricht man von Grafik, muß man zwangsläufig auch auf die hardwarebedingten Grafikfähigkeiten eingehen. Wenn Sie einen Blick auf Ihre Basic-Systemdiskette werfen, so werden Sie außer GW-Basic unter Umständen noch eine weitere Basic-Version mit dem Namen »hbasic.exe« finden. Dieses Programm ist eine spezielle Basic-Version für Personalcomputer mit Hercules-Karte. Die Unter-

schiede werden aber erst im zweiten Grafikmodus sichtbar.

Während dort die Auflösung mit CGA-Grafik nur 640x200 Punkte beträgt, stellt HBasic dem Benutzer mit Hercules-Karte 720x348 Bildpunkte zur Verfügung. Tabelle 1 gibt einen kleinen Überblick über die wichtigsten Unterschiede zwischen GW-Basic und HBasic. HBasic erkennt dabei selbstständig, ob das Computersystem mit einer Hercules-Karte und dem entsprechenden Monitor ausgerüstet ist. Trifft dies nicht zu, so wird der Ladevorgang mit einer entsprechenden Fehlermeldung abgebrochen.

Der Color-Befehl zur Einstellung der Farben ist nicht nur den Anwendern mit Farbmonitoren vorbehalten. Betreiben Sie Ihren Computer an einem monochromen Bildschirm, bleibt immer noch die Wahl zwischen Weiß, Schwarz und Intensiv-Weiß. Seine ganze »Farbenpracht« entlockt Ihrem Computer dann allerdings nur der Textmodus. Dort stehen Ihnen nämlich 16 verschiedene Zeichenfarben (0 bis 15), 8 Hintergrundfarben (0 bis 7) und 16 Rahmenfarben (0 bis 15) zur Verfügung. Im Textmodus (»screen 0«) hat der Color-Befehl folgende Syntax:

color zeichenfarbe, hintergrundfarbe, rahmenfarbe

Im Grafikmodus ist alles etwas anders oder: geänderte Syntax bei Basic-Befehlen

In Tabelle 2 werden die 16 Farbnummern mit ihrer Wirkung am Farb- und Monochrom-Monitor aufgeführt. Alle Parameter der Color-Anweisung sind »optional«; das heißt, es dürfen beliebige Parameter einfach entfallen. Soll nur die Zeichenfarbe verändert werden, so reicht der Befehl »color zeichenfarbe«. Wollen Sie nur die Hintergrundfarbe auf Kobaltblau set-

zen, so müssen Sie den Befehl »color ,3« eingeben. Die übrigen Parameter behalten ihren Wert.

Der Parameter für die Zeichenfarbe darf Werte zwischen 0 und 31 annehmen. Wird hier zur Zeichenfarbe der Wert 16 hinzuaddiert, so erscheint das Zeichen in derselben Farbe, jedoch blinkend. Die beiden anderen Parameter dürfen nur die angegebenen Farbwerte annehmen. Der Blinkmodus ist dort nicht zugelassen.

Im Grafikmodus hat der Color-Befehl eine andere Syntax. Hier sind nur noch zwei Parameter vorgesehen:

color hintergrundfarbe, palette

In diesem Fall darf der Parameter »hintergrundfarbe« einen Wert zwischen 0 und 15 annehmen. Auch hier ist die Tabelle 2 anwendbar. Für »palette« dürfen die Zahlen 0 bis 255 eingesetzt werden. Entscheidend ist nur die Tatsache, ob der Wert gerade oder ungerade ist. Bei ungeradem Parameter wird die Farbpalette 1 benutzt, andernfalls Palette 0. Jede Farbpalette enthält drei Farben, die über die Kennziffern 1, 2 und 3 aufgerufen werden. Die Kennziffer 0 übergibt die aktuelle Hintergrundfarbe. Standardmäßig ist Palette 0 mit folgenden Farben belegt:

- 0 = aktuelle Hintergrundfarbe
- 1 = Grün
- 2 = Rot
- 3 = Braun

Palette 1 weist dagegen diese vier Farben auf:

- 0 = aktuelle Hintergrundfarbe
- 1 = Kobaltblau
- 2 = Violett
- 3 = Weiß

Natürlich muß sich der Programmierer nicht nur auf diese Farben beschränken. Werden andere Farben benötigt, so können Sie die aktuelle Farbpalette umprogrammieren. Hier-

für ist GW-Basic mit dem Befehl »palette« ausgestattet:

palette kennziffer, farbnnummer

»kennziffer« wählt die zu ändernde Farbe der Palette, während der Parameter »farbnnummer« die neugewählte Farbe (nach Tabelle 2) enthalten muß. Wird der letztgenannte Parameter mit »-1« angegeben, so erfolgt keine Änderung der Farbe. Mit Hilfe dieses Befehls läßt sich jeweils nur eine Farbe einer Palette verändern. Sollen gleichzeitig alle vier Farben neu definiert werden, so wird dem Palette-Befehl das Schlüsselwort »using« angehängt. Die vollständige Syntax für diesen Fall muß wie folgt lauten:

palette using farbe%

Der Parameter »farbe%« muß eine Integer-Feldvariable sein, welche die vier neuen Farben der Palette in aufeinanderfolgender Reihenfolge enthält. Folgendes Beispiel-Listing definiert die aktuelle Farbpalette mit den Farben Rot, Grün und Blau. Die Hintergrundfarbe wird dabei auf Weiß gesetzt.

```
100 rem *****
110 rem * Palette-Using-Demo *
120 rem *****
130 :
140 for i=0 to 3
150 read farbe%(i)
160 next i
170 :
180 palette using farbe%(0)
190 :
200 data 7,4,2,1
210 rem Weiß,Rot,Grün,Blau
```

Werden mit den Befehlen »palette« oder »palette using« Farbbänderungen vorgenommen, so erfolgen diese ebenfalls am gesamten Bildschirm. Wird also die Farbe Rot der aktuellen Farbpalette in Blau verändert, so erscheinen augenblicklich alle ursprünglich roten Bildpunkte in blauer Farbe. Außerdem können bei mitt-

lerer CGA-Auflösung niemals mehr als vier verschiedene Farben gleichzeitig auf dem Bildschirm ausgegeben werden.

Mit steigender Zahl der Bildpunkte sinkt im gleichen Verhältnis die Farbvielfalt

Wurde mit dem Befehl »screen 2« die CGA-Karte in die hohe Grafikauflösung von 640x200 Bildpunkten geschaltet, so stehen dem Programmierer gerade noch die beiden Farben Schwarz und Weiß zur Auswahl. Soll ein Bildpunkt in Weiß gesetzt werden, brauchen Sie dazu die Farbnnummer 1. Im anderen Falle wird die Farbnnummer auf 0 gesetzt.

GW-Basic kennt viele Befehle, die den Aufbau von Grafiken erleichtern (Tabelle 3). Ein sehr vielfältiger Befehl ist dabei die Circle-Anweisung. Sie erlaubt das Zeichnen von Kreisen und Ellipsen in allen erdenklichen Variationen. Die vollständige Syntax dieses Befehls lautet:

circle (x,y),radius,farbe,anfang,ende,verhältnis

Die Mittelpunktswinkelkoordinaten des Kreises werden durch die Parameter »x« und »y« übergeben. »radius« enthält den Kreisradius in x-Richtung. Es wird nur deshalb zwischen x- und y-Richtung differenziert, weil in den meisten Grafikmodi eine Gleichverteilung von x- und y-Radius nur eine Ellipse und keinen Kreis erzeugen würde. Lediglich HBasic bildet am monochromen Monitor in der Auflösung 720x348 Bildpunkte einen echten Kreis ab.

Stellt Ihr Bildschirm keinen Kreis dar, so müssen Sie das Verhältnis der Koordinatenachsen anpassen. Hierfür steht Ihnen der Parameter »verhältnis« zur Verfügung. »verhältnis« paßt, entsprechend seinem Wert,

Befehl	Erklärung
CIRCLE	zeichnet einen Kreis oder eine Ellipse
CLS	löscht den aktuellen Bildschirm
COLOR	setzt Farbe für Zeichen, Hintergrund und Rahmen
DRAW	zeichnet ein Grafikobjekt
GET	liest ein Shape in eine Feldvariable ein
LINE	zeichnet eine Linie oder ein Viereck
PAINT	füllt ein Grafikobjekt mit einer Farbe
PALETTE	ändert die Farbe einer Farbpalette
PALETTE USING	ändert alle Farben einer Farbpalette
PMAP	rechnet Grafikkoordinaten um
POINT	übergibt Farbe und Koordinaten des Bildpunkt-Cursors
PRESET	setzt einen Grafikpunkt
PSET	setzt einen Grafikpunkt
PUT	setzt ein Shape aus einer Feldvariablen in die Grafik
SCREEN	setzt wichtige Bildschirm-Attribute
VIEW	definiert einen Grafikbereich als »Window«
WINDOW	legt Window-Ausgabe fest

Tabelle 3. Alle Grafikbefehle auf einen Blick

Kayser Daten-Technik

J. Kayser Gladiolenweg 2a 4796 Salzkotten
Telefon 0 52 58/30 63 Telefax 0 52 58/42 98

Software für IBM* Personalcomputer
XT – AT – PS/2

Professionell in Qualität und Preis

Neu! Neu! Neu! Neu! Neu!

MIP-Kasse	Kassenbuch und Einnahme-/Uberschubrechnung für Gewerbe und Privatpersonen, täglicher Kassenstand, freie Kontenverwaltung, USt-Listen, Journal, Taschenrechner. 1 Diskette 5,25 Zoll, 1 Diskette 3,5 Zoll, Handbuch	69,-
MIP-Büro	Internationales Büropaket mit Rechnungswesen, Lager, Versand, Postvertrieb, Buchhaltung, Mahnwesen, offene Posten, Präsentationsgrafik, Textverarbeitung, Termine, Stammdaten, freie Steuersätze, freie Formulargestaltung, freiwählbare Währung. 1 Diskette 5,25 Zoll, 1 Diskette 3,5 Zoll, Handbuch	299,-
MIP-Text Plus	Textverarbeitung und Programmeditor, Zeilenumbruch, Blocksatz, Makros, Suchfunktion, Serienbriefe, Adresskartei mit Notizblatt je Adresse, Sortier Routinen auf Bildschirm und Drucker. 1 Diskette 5,25 Zoll, 1 Diskette 3,5 Zoll, Handbuch	69,-
MIP-Druck Plus	Programm zum Erstellen beliebiger Etiketten mit verschiedenen Schriftarten und Seriendruck, speicherbar, Formulardruck, Überweisungen, Scheckdruck. 1 Diskette 5,25 Zoll, 1 Diskette 3,5 Zoll, Handbuch	69,-
MIP-Post	Dieses Programm verwaltet Ihren kompletten Postvertrieb mit Druck von Paketkarten, Nachnahme-Karten, Briefnachnahme, Paketaufkleber, Adresskartei Notizen. 1 Diskette 5,25 Zoll, 1 Diskette 3,5 Zoll, Handbuch	69,-
MIP-Aktie	Programm zum Verwalten von Aktien und Optionsscheinen in 9 verschiedenen Depots mit Depotbestand, Gewinn-/Verlustrechnung, aktuelle Kurse, Druckroutine. 1 Diskette 5,25 Zoll, 1 Diskette 3,5 Zoll, Handbuch	69,-
MIP-Schule	Programm zum Verwalten von Schülerstammdaten mit Sortier Routinen auf Bildschirm und Drucker, Klassenliste, Fahr Schülererfassung, Schullaufbahn, Schulabgang. 1 Diskette 5,25 Zoll, 1 Diskette 3,5 Zoll, Handbuch	69,-
MIP-Maus	Jede Software läuft über dieses Programm mit der Maus (z.B. dBase, Framework)*. Die Empfindlichkeit der Maus ist frei wählbar, speicherresistent	29,-
Maus-Set	Genius GM 6 plus Maus und MIP-Maus incl. Grafikprogramm und Mauspad	120,-
MIP-Fakt	Komplettes Fakturierungsprogramm mit Kunden-/Lieferantenkartei, Rechnungen, Mahnungen, Buchhaltung, Statistik mit Präsentationsgrafik, Datensicherung	29,-
MIP-Lager	Lagerverwaltungsmodul für MIP-Fakt; Artikel können direkt in die Rechnung übernommen werden; Preislisten, Inventar, Artikelaufkleber, Einlagen	29,-
MIP-Dos	Lernprogramm für die wichtigsten Dos-Befehle und deren Einsatz	29,-
MIP-Term	Terminplaner mit Adresskartei, Sortier Routinen auf Bildschirm und Drucker, Etiketten und Notizblatt je Adresse	29,-
Diskettenbox	incl. 10 Disk 3,5 Zoll 2 DD	45,-
Diskettenbox	incl. 10 Disk 5,25 Zoll 2 D	29,-
Diskettenbox	incl. 5,25 Zoll 2 HD	45,-

Handleranfragen erwünscht *eingetragene Warenzeichen

den y-Radius dem vorgegeben x-Radius an. Erfahrungswerte für die CGA-Grafikauflösung liegen bei einem Wert von (5/6). In der hohen Auflösungsstufe sollten Sie den Wert (5/12) verwenden, um echte Kreise zu erhalten.

Listing 2 gibt in einer For-Next-Schleife 1000 Zufallskreise auf dem Bildschirm aus. Zusätzlich wird jeder zweite Kreis durch Ausmalen in eine Scheibe verwandelt. Ein Blick auf das kurze Listing läßt einige interessante Grafikbefehle erkennen. Zunächst fällt der Screen-Befehl ins Auge. Durch die Anweisung in Zeile 140 wird auf die maximale Auflösung umgeschaltet. Die Screen-Anweisung verarbeitet bis zu vier Parameter:

```
screen modus, farbe, aktuelle_seite,
sichtbare_seite
```

Folgende Modi sind für den Parameter »modus« zulässig:

- »0« – Textmodus mit 40 oder 80 Zeichen pro Bildschirmzeile
- »1« – mittlere CGA-Grafikauflösung mit 320x200 Bildpunkten
- »2« – hohe CGA-Grafikauflösung mit 640x200 Bildpunkten (GW-Basic) oder Hercules-Grafik mit 720x348 Bildpunkten (HBasic)

Der zweite Parameter »farbe« hat den Charakter einer booleschen Variablen. Ist »farbe« gleich 1 (logisch wahr), ist keine Farbdarstellung zugelassen. Dies ist nur dann der Fall, wenn »farbe« den Wert 0 (logisch falsch) annimmt. Dieser Parameter hat natürlich nur in den Modi 0 und 1 seine Berechtigung, da im hochauflösenden Grafikmodus sowieso nur Schwarzweiß-Darstellung erlaubt ist. Die Parameter »aktuelle_seite« und »sichtbare_seite« dürfen nur im Textmodus benutzt werden. Mit »aktuelle_seite« wird die zu bearbeitende Bildschirmseite ausgewählt. Das muß nicht unbedingt die gerade sichtbare Bildschirmseite sein. Diese wird nämlich durch den letzten Parameter bestimmt. Es ist somit denkbar, eine Bildschirmseite zu bearbeiten, während eine andere am Bildschirm angezeigt wird.

Beinahe schon symptomatisch für Grafikbefehle sind die mannigfaltigen Parameter

Je nach Darstellung im Textmodus lassen sich bis zu acht verschiedene Bildschirmseiten (0 bis 7) bearbeiten. Dieses Maximum von acht Seiten ist jedoch nur bei einer Darstellung von 40 Zeichen pro Textzeile möglich. Werden 80 Zeichen pro Zeile verwendet, halbiert sich die Anzahl der verfügbaren Bildschirmseiten auf vier (0 bis 3).

Zurück zum Beispielpogramm (Listing 2): Die Cls-Anweisung in Zeile 150 löscht den Grafikbildschirm und positioniert den Cursor am linken oberen Bildschirmrand. In der For-Next-Schleife zwischen den Zeilen 210 und 310 werden zunächst Zufallswerte für den Radius, die Farbe sowie x- und y-Position ermittelt. Anschließend wird der Kreis mit den entsprechenden Attributen am Bildschirm dargestellt. Sollten Sie an Ihrem Bildschirm nur Ellipsen sehen, so muß die Zeile 270 um den Parameter »verhältnis« erweitert werden. Versuchen Sie es zunächst mit dem Wert (5/12), weil dieser in der Regel zum Erfolg führt. Nur wenn Ihr Bildschirm ein sehr exotisches Format besitzt, müssen Sie mit dem Parameter etwas experimentieren.

In Zeile 290 überprüft das Programm, ob der Zähler »i« eine gerade Zahl darstellt. Ist dies der Fall, wird der soeben gezeichnete Kreis ausgefüllt. Dieser kleine Trick soll bewirken, daß nur jeder zweite Kreis mit dem Paint-Befehl bearbeitet wird. Auch dieser Befehl verarbeitet bis zu fünf Parameter:

```
paint (x,y), füllfarbe, grenzf Farbe,
hintergrund
```

Ähnlich wie beim Circle-Befehl geben die Parameter »x« und »y« den Startbereich für den Ausmalvorgang an. Der Parameter »füllfarbe« gibt in der Regel die gewünschte Füllfarbe wieder. Bei mittlerer Grafikauflösung sind Werte zwischen 0 und 3, bei hoher Auflösung nur die Werte 0 und 1 erlaubt.

Der Parameter »füllfarbe« ruft jedoch noch weitere Effekte hervor. Wird anstelle des Farbwertes eine String-Variable angegeben, so füllt »paint« den angegebenen Bereich mit einem Füllmuster. Diese String-Variable darf aus maximal 64 Byte bestehen, wobei jedes Bit einem Bildpunkt entspricht. Der Parameter »grenzf Farbe« enthält den Farbcoder, der die zu füllende Fläche umschließt. Wird der Parameter nicht angegeben, so setzt GW-Basic automatisch Füllfarbe gleich Grenzf Farbe. Der Parameter »hintergrund« ist im gleichen Sinne eine String-Variable, die das Bitmuster des zu füllenden Bereichs enthält (natürlich nur innerhalb der Grenzen). Standardmäßig wird für diesen Parameter »chr\$(0)« angenommen.

Doch nun zurück zur Circle-Anweisung. Zum besseren Verständnis der Parameter »anfang« und »ende« wurde noch ein kleines Beispielpogramm (Listing 3) entworfen. Das Programm wandelt zehn Benutzereingaben in ein Kuchendiagramm um (Bild 2).

Die Parameter »anfang« und »ende« sind numerische Variablen, die

Anfangs- und Endwinkel eines Kreisausschnittes festlegen. Beide müssen im Bogenmaß angegeben werden. Die Umrechnung von Grad in Bogenmaß ist mit folgender Formel leicht zu bewerkstelligen:

Bogenmaß = Grad * Pi/180

Werden negative Winkelangaben verwendet, so verbindet GW-Basic Anfangs- und Endpunkt des Kreisbogens mit dem Kreismittelpunkt. Diese Tatsache nutzt auch das Beispielpogramm »Kuchendiagramm«. Dadurch, daß der Anfangswinkel in Zeile 410 auf »-2 Pi« gesetzt wurde, werden alle Kreissegmente als »Kuchenstücke« dargestellt. Probieren Sie ruhig einmal andere Anfangswerte für Ihr Kuchendiagramm. Geben Sie beispielsweise Null als Anfangswert vor, so sehen Sie als Grafik lediglich einen Kreisbogen.

Zur Funktion des Programms genügen einige kurze Erklärungen. In den Zeilen 240 bis 280 werden zehn Werte eingelesen und der Maximalwert berechnet. Die For-Next-Schleife ab Zeile 340 stellt die zehn zugehörigen Prozentwerte dar und berechnet die zehn Teilwinkel im Bogenmaß. Die Ausgabe der Kuchengrafik ist schließlich in den Zeilen 440 bis 480 realisiert.

Linien und Kästchen zu zeichnen, ist mit dem Line-Befehl ganz einfach

Auch die Line-Anweisung ist ein vielfältiges Grafikwerkzeug. Damit lassen sich sowohl Linien als auch Rechtecke zeichnen. Bei Bedarf werden die Kästchen mit der aktuellen Zeichenfarbe ausgemalt. Die Line-Anweisung besitzt folgende Syntax:

```
line (x1,y1)-(x2,y2), farbe, b < f>, maske
```

Die Parameter »x1«, »x2«, »y1« und »y2« enthalten die Bildschirmkoordinaten zweier Bildpunkte. Sie haben je nach vorhandenem Vorzeichen absoluten oder relativen Charakter. Im Normalfall zeichnet die Anweisung eine Linie in der gewählten Zeichenfarbe von Punkt (x1,y1) nach (x2,y2). Durch Einsetzen der Marke »b« (kein Parameter) wird anstelle der Linie ein Viereck über die beiden Punkte gespannt. Folgt dem »b« noch der Buchstabe »f«, so füllt der Befehl das gezeichnete Rechteck mit der aktuellen Zeichenfarbe. Wünschen Sie keine durchgezogenen Linien, so können Sie die Form der Linie mittels einer 16-Bit-Maske verändern. Diese Maske läßt sich am besten mit einer hexadezimalen Konstante definieren. Eine gestrichelte Linie ist beispielsweise mit der Maske &HCCCC zu erreichen. Dies entspricht einer Binärzahl, bei der sich jeweils zwei

Nullen und Einsen abwechseln (1100110011001100).

Listing 4 zeigt ein kleines Demo-Programm, das sowohl die verschiedenen Varianten des Line-Befehls, als auch die Füll-Option der Paint-Anweisung demonstriert. In den Zeilen 320 bis 370 werden ab Position (0,0) größer werdende, ausgemalte Rechtecke dargestellt. Wenn Sie die Zeile 360 aufmerksam betrachten, werden Sie feststellen, daß die erste Koordinatenangabe fehlt. Auch diese Schreibweise ist korrekt, da GW-Basic in diesem Fall den letzten Grafikpunkt als Startkoordinate einsetzt. Wird eine der beiden Koordinaten nicht angegeben, so setzt GW-Basic automatisch die letzte Cursor-Position ein. In Zeile 390 wird ein Rahmen um den gesamten Bildschirm gezeichnet. Dies ist für die folgende Paint-Anweisung erforderlich. Aus demselben Grund umrandet ein Kreis den Titel (Zeile 410).

Doch nun zurück zur Paint-Anweisung. Wie Sie inzwischen wissen, darf der Parameter »füllfarbe« einen bis zu 64 Byte langen String-Ausdruck beinhalten. Jedes Byte dieser Zeichenkette entspricht acht horizontalen Bildpunkten. Die einzelnen Bytes hingegen werden untereinander dargestellt. Als Füllmuster wurde der Schriftzug »DOS« verwendet. Insgesamt benötigen Sie 25 Byte für dieses Zeichenraster. Die zugehörigen Codierungen sind in den Data-Zeilen (Zeile 480 und 490) untergebracht. In den Zeilen 200 bis 230 wird die String-Variable »fuell\$« aus den Data-Elementen generiert. Die Paint-Anweisung in Zeile 430 füllt die rechte obere Ecke des Bildschirms mit dem neu definierten Füllmuster.

Der Draw-Befehl ist wegen seiner zahlreichen Parameter sehr leistungsfähig

Der mit Abstand leistungsfähigste Grafikbefehl von GW-Basic ist »draw«. Mit ihm lassen sich wegen seiner integrierten »Steueranweisungen« auf einfachste Weise komplexe Objekte zeichnen (Bild 3). »draw« weist folgende Syntax auf:

draw zeichenkette

Die Zeichenkette enthält die sogenannten Steueranweisungen oder Zeichenkommandos des Befehls. Zur Demonstration der Fähigkeiten des Draw-Befehls dient das letzte Beispielsprogramm (Listing 5). In den Zeilen 490 bis 680 wird der Schriftzug »DOS« als positionsunabhängige Grafik am Bildschirm dargestellt. Die Steueranweisungen beschränken sich ausschließlich auf Bewegungsbefehle. Lediglich im letzten Programmteil (ab Zeile 730) wird die C-Anweisung

zur negativen Darstellung des Schriftzuges benötigt. Um das Demo-Programm optisch aufzubereiten, wird der Schriftzug in größer werdenden Schritten als Sinus-Funktion angezeigt. Tabelle 4 führt alle zulässigen Zeichenkommandos des Draw-Befehls auf.

Indem Sie den Buchstaben »b« vor einen Bewegungsbefehl (»u«, »d«, »l«, »r«, »e«, »f«, »g«, »h« oder »m«) stellen, wird während der Cursorbewegung nicht gezeichnet. Wird der Buchstabe »n« vorangestellt, so bleibt die Ausgangsposition der Bewegung trotz des verschobenen Cursors erhalten. Die Steueranweisung »a« läßt nur die Werte 0 bis 3 zu. Dabei gilt folgende Zuordnung:

- »0« – Winkel von 0 Grad
- »1« – Winkel von 90 Grad
- »2« – Winkel von 180 Grad
- »3« – Winkel von 270 Grad

Das Zeichenkommando »m x,y« besitzt zwei Ausführungsvariationen: Wird den Parametern »x« und »y« kein Vorzeichen vorangestellt, so beziehen sich die Angaben auf die absolute Koordinatenposition. Die Angabe eines Vorzeichens verschiebt die Punkte relativ zum letzten Endpunkt. Liegt dem Anwender eine fertige Grafik vor, so ist es oft wünschenswert, die Attribute eines bestimmten Bildpunktes abzufragen. Hierfür bietet GW-Basic den Point-Befehl mit folgender Syntax:

point (x,y)
point (n)

Mit »point (x,y)« wird zunächst der gewünschte Punkt selektiert. Anschließend erfragen Sie mit »point (n)« beliebige Attribute:

- n – Attribut
- 0 – liefert aktuelle X-Koordinate
- 1 – liefert aktuelle Y-Koordinate
- 2 – liefert aktuelle X-Anwender-Koordinate (bei Window)
- 3 – liefert aktuelle Y-Anwender-Koordinate (bei Window)

Auch dieser Befehl findet im Bei-

spielsprogramm (Listing 5) seine Verwendung. Wie Sie aus dem Programm ersehen, werden mehrmals die aktuellen Bildschirmkoordinaten abgefragt, um Sie entsprechend zu verändern und neu zu setzen. Hierfür stehen in GW-Basic die gleichwertigen Befehle »preset« und »pset« zur Verfügung. Mit dieser Anweisung läßt sich der Grafik-Cursor an jede beliebige Bildschirmposition setzen. Die Syntax dieser Befehle lautet:

preset (x,y),farbe
pset (x,y),farbe

GW-Basic erlaubt die Programmierung sogenannter Shapes. Shapes sind Grafikausschnitte, die einem vollständigen Grafikbild entnommen und an anderer Stelle oder anderen Stellen wieder eingefügt werden. Mit Shapes läßt sich beispielsweise bedingt Animationen simulieren. Der Befehl »get« dient zur Übergabe eines Bildschirmausschnittes an eine Feldvariable. Die Syntax dieses Befehls lautet:

get (x1,y1)-(x2,y2),feldvariable

GW-Basic unterstützt den Programmierer beim Arbeiten mit Grafikausschnitten

Eine Erklärung dieses Befehls erübrigt sich, weil die Koordinatenangaben identisch mit denjenigen des Line-Befehls sind. Interessant ist lediglich die Berechnung des Platzbedarfes eines Shapes, wenn die relativen Ausmaße bekannt sind:

Bytes = Int((Bitwertigkeit*x+7)/8)*Y

Bei hoher Grafikauflösung beträgt die Bitwertigkeit 1, andernfalls 2. Wiedergegeben wird der gespeicherte Bildschirmausschnitt durch die Put-Anweisung:

put (x,y),feldvariable,modus

Auch dieser Befehl erklärt sich bis auf den Parameter »modus« von selbst. Modus beschreibt die Wiedergebart des Shapes (Tabelle 5). Oft ist

Tabelle 4. Eine Zusammenstellung der Steueranweisungen des Draw-Befehls

Befehl	Erklärung
U n	zeichnet n Bildpunkte nach oben
D n	zeichnet n Bildpunkte nach unten
L n	zeichnet n Bildpunkte nach links
R n	zeichnet n Bildpunkte nach rechts
E n	zeichnet n Bildpunkte nach rechts oben
F n	zeichnet n Bildpunkte nach rechts unten
G n	zeichnet n Bildpunkte nach links unten
H n	zeichnet n Bildpunkte nach links oben
M x,y	setzt Cursor an (x,y)-Position
A n	setzt einen rechten Winkel (0, 90, 180 oder 270 Grad)
TA n	setzt beliebigen Winkel (-360 bis 360 Grad)
C n	setzt die Farbe n
S x	legt Skalenfaktor zwischen 1/4 und 255/4 fest
X v\$	führt Befehls-String aus
P n,m	füllt Objekt mit einer Farbe aus

es wünschenswert, daß Manipulationen an einem Shape oder am Bildschirmhintergrund vorgenommen werden, bevor das Shape wiedergegeben wird. Soll die grafische Ausgabe nur in einem bestimmten Bildschirmbereich erfolgen, so läßt sich dieser mit Hilfe der View-Anweisung selektieren. Die vollständige Syntax der View-Anweisung lautet:

```
view screen (x1,y1)-(x2,y2),farbe,randfarbe
```

Die Eckpunkte des Bildschirmbereiches dürfen sowohl absolut als auch relativ angegeben werden. Interes-

sant ist der Parameter »randfarbe«, der dem Benutzer erlaubt, den Bildschirmbereich mit einer beliebigen Farbe zu umrahmen. Hinsichtlich der Anzahl von Bildschirmfenstern sind programmtechnisch keine Grenzen gesetzt. Ist ein bestimmter Bildschirmausschnitt aktiviert, dann wirkt der Befehl »cls« nur in diesem Bereich des Schirmes.

Wie Sie sehen, ist GW-Basic mit erstaunlich vielen Grafikbefehlen ausgerüstet. Es dürfte Ihnen somit nicht mehr schwerfallen, geeignete Grafik-anwendungen zu programmieren.

(Christian Spitzner/zi)

Modus	Bedeutung
PSET	Shape wird originalgetreu dargestellt
PRESET	Shape wird invertiert dargestellt
AND	Shape wird mit dem Hintergrund logisch Und-verknüpft
OR	Shape wird mit dem Hintergrund logisch Oder-verknüpft
XOR	Shape wird mit dem Hintergrund logisch Exklusiv-Oder-verknüpft

Tabelle 5. Die Wiedergabe-Modi des Shape-Befehls »put«

```
[885] 1 REM Programmname: GRAFIK1.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[FR0] 4 REM
[TD4] 100 REM *****
[MW6] 110 REM * Menügesteuerte Erklärung der *
[SK5] 120 REM * Grafik-Befehle (sp011288) *
[LE4] 125 REM *****
[FK0] 130 :
[P85] 140 DIM MENUES$(40),HINTERGRUND$(25)
[0D2] 150 FOR I=1 TO 25
[CH5] 160 HINTERGRUND$(I) = STRING$(80," ")
[A67] 165 'Hintergrund-Variablen initialisieren
[1H1] 170 NEXT I
[YK0] 180 :
[AY1] 190 LOCATE , ,0
[5U1] 200 KEY OFF
[BC2] 210 FOR I=1 TO 10
[RR1] 220 KEY I,""
[RG1] 230 NEXT I
[HY1] 240 GOSUB 1920
[N76] 245 'Initialisiert das Pull-Down-Menü
[XY1] 250 GOSUB 1680
[924] 255 'Holt Menü-Parameter
[UK0] 260 :
[F04] 270 REM * Hauptprogramm *
[0L0] 280 :
[LM1] 290 WHILE 1
[U52] 300 GOSUB 1290
[P17] 305 'Pull-Down-Menü-Verwaltungsprogramm
[R31] 310 CLS
[UP1] 320 PRINT
[7H6] 330 PRINT "-----"
[GG6] 340 PRINT "-----"
[---] 350 :
[D42] 350 COLOR 0,7
[4W8] 360 PRINT TAB(35)MENUES$(STARTSTRING(MENUE)+BEFEHL-1);
[3W2] 370 PRINT TAB(80);
[842] 380 COLOR 7,0
[TH6] 390 PRINT "-----"
[6G6] 400 PRINT "-----"
[---] 410 :
[TP1] 410 PRINT
[7E4] 420 ON MENUE GOSUB 480,530,600
[Z52] 430 GOSUB 5070
[2H4] 435 'Wartet auf Tastendruck
[X61] 440 WEND
[VK0] 450 :
[E94] 460 REM * Pull-Down-Menü 1 *
[1L0] 470 :
[LR4] 480 ON BEFEHL GOSUB 2360,2440,2450
[TM1] 490 RETURN
[EK0] 500 :
[494] 510 REM * Pull-Down-Menü 2 *
[MK0] 520 :
[AY8] 530 ON BEFEHL GOSUB 2490,2640,2680,2760,2850,3010,3140,3240,3330,3410
[CB4] 540 IF BEFEHL<11 THEN RETURN
[BE7] 550 ON BEFEHL-10 GOSUB 3530,3660,3740,3820,3940,4060,4150
[KM1] 560 RETURN
[3L0] 570 :
[W94] 580 REM * Pull-Down-Menü 3 *
[9L0] 590 :
[M28] 600 ON BEFEHL GOSUB 4270,4310,4350,4390,4430,4470,4510,4550,4590,4630
[5B4] 610 IF BEFEHL<11 THEN RETURN
```

```
[GA6] 620 ON BEFEHL-10 GOSUB 4710,4790,4870,4930,5000
[BM1] 630 RETURN
[WK0] 640 :
[ZK0] 650 :
[BN6] 660 REM * Hintergrundinformation retten *
[5L0] 670 :
[AZ9] 680 FOR I=FENSTERSPALTE TO FENSTERSPALTE + FENSTERLAENGE - 1
[A89] 690 FOR J=STARTSPALTE TO STARTSPALTE + FENSTERBREITE - 1
[PF7] 700 MID$(HINTERGRUND$(I),J,1)=CHR$(SCREEN(I,J))
[ML1] 710 NEXT J
[YG1] 720 NEXT I
[DM1] 730 RETURN
[YK0] 740 :
[2F8] 750 REM * Hintergrundinformation zurückschreiben *
[4L0] 760 :
[1N9] 770 FOR I=FENSTERSPALTE TO FENSTERSPALTE+FENSTERLAENGE-1
[8SD] 780 LOCATE I,STARTSPALTE:PRINT MID$(HINTERGRUND$(I),STARTSPALTE,FENSTERBREITE);
[LH1] 790 NEXT I
[6M1] 800 RETURN
[RK0] 810 :
[RP5] 820 REM * Pull-Down-Menü zeichnen *
[XK0] 830 :
[P55] 840 LOCATE 2,STARTSPALTE(MENUE)
[NE7] 850 PRINT "r"+STRING$(BREITE(MENUE),"-")+ "j"
[4C4] 860 FOR I=1 TO LAENGE(MENUE)
[B55] 870 LOCATE ,STARTSPALTE(MENUE)
[528] 880 PRINT " |"+MENUES$(STARTSTRING(MENUE)+I-1)+" |"
[NH1] 890 NEXT I
[WX4] 900 LOCATE ,STARTSPALTE(MENUE)
[TG7] 910 PRINT "l"+STRING$(BREITE(MENUE),"-")+ "j"
[EM1] 920 RETURN
[ZK0] 930 :
[EM4] 940 REM * Option auswählen *
[5L0] 950 :
[ZG6] 960 LOCATE BEFEHL2+2,STARTSPALTE(MENUE)+1
[2K7] 970 PRINT MENUES$(STARTSTRING(MENUE)+BEFEHL2-1)
[EL0] 980 :
[HG6] 990 LOCATE BEFEHL+2, STARTSPALTE(MENUE)+1
[MX1] 1000 COLOR 0,7
[DG7] 1010 PRINT MENUES$(STARTSTRING(MENUE)+BEFEHL-1)
[EX1] 1020 COLOR 7,0
[4V1] 1030 RETURN
[0R0] 1040 :
[NM5] 1050 REM * Aktuelles Menü anzeigen *
[8R0] 1060 :
[X96] 1070 STARTSPALTE=STARTSPALTE(MENUE2)
[183] 1080 FENSTERSPALTE=2
[8R5] 1090 FENSTERBREITE=BREITE(MENUE2)+2
[2M5] 1100 FENSTERLAENGE=LAENGE(MENUE2)+2
[KX1] 1110 GOSUB 770
[SB5] 1120 LOCATE 1,STARTSPALTE(MENUE2)
[N1A] 1130 PRINT MID$(KOPF$,STARTSPALTE(MENUE2),NAMENLAENGE(MENUE2))
[776] 1140 STARTSPALTE=STARTSPALTE(MENUE)
[S73] 1150 FENSTERSPALTE=2
[904] 1160 BREITE=BREITE(MENUE)
[PW3] 1170 LAENGE=LAENGE(MENUE)
[9Y1] 1180 GOSUB 840
[VY1] 1190 COLOR 0,7
[S85] 1200 LOCATE 1,STARTSPALTE(MENUE)
[YS9] 1210 PRINT MID$(KOPF$,STARTSPALTE(MENUE),NAMENLAENGE(MENUE))
[3N6] 1220 LOCATE BEFEHL+2,STARTSPALTE(MENUE)+1
[VG7] 1230 PRINT MENUES$(STARTSTRING(MENUE)+BEFEHL-1)
[WX1] 1240 COLOR 7,0
[KV1] 1250 RETURN
```



```
[ER0] 1260 :
[FG4] 1270 REM * Menü-Verwaltung *
[PR0] 1280 :
[U03] 1290 STARTSPALTE=1
[973] 1300 FENSTERSPALTE=1
[5A3] 1310 FENSTERBREITE=80
[AG4] 1320 FENSTERLAENGE=MAXIMAL+3
[1Y1] 1330 GOSUB 680
[246] 1335 'Hintergrundinformation retten
[U72] 1340 LOCATE 1,1
[DF2] 1350 PRINT KOPF$
[CR4] 1355 'Menü-Leiste darstellen
[AT1] 1360 MENUE=1
[SW1] 1370 BEFEHL=1
[MV1] 1380 MENUE2=1
[T42] 1390 GOSUB 1070
[WP0] 1400 :
[2R2] 1410 REM * Eingabe *
[4R0] 1420 :
[M31] 1430 A$=""
[873] 1440 WHILE A$<>CHR$(13)
[2X2] 1450 MENUE2=MENUE
[MY2] 1460 BEFEHL2=BEFEHL
[BP5] 1470 A$=INKEY$:IF A$="" THEN GOTO 1470
[NP7] 1480 IF LEN(A$)>2 THEN GOTO 1530 ELSE A$=RIGHT
$(A$,1)
[BPA] 1490 IF ASC(A$)=77 THEN MENUE=MENUE+1: IF MEN
UE>ANZAHL THEN MENUE=1
[YMA] 1500 IF ASC(A$)=75 THEN MENUE=MENUE-1: IF MEN
UE<1 THEN MENUE=ANZAHL
[YYB] 1510 IF ASC(A$)=72 THEN BEFEHL=BEFEHL-1: IF BEF
EHL<1 THEN BEFEHL=LAENGE(MENUE)
[AZB] 1520 IF ASC(A$)=80 THEN BEFEHL=BEFEHL+1: IF BEF
EHL>LAENGE(MENUE) THEN BEFEHL=1
[3V5] 1530 IF BEFEHL<>BEFEHL2 THEN GOSUB 960
[FC7] 1540 IF MENUE <> MENUE2 THEN BEFEHL=1:GOSUB 107
0
[M91] 1550 WEND
[RR0] 1560 :
[KM8] 1570 REM * Hintergrundinformation wiederherstelle
n *
[ZR0] 1580 :
[313] 1590 STARTSPALTE=1
[K73] 1600 FENSTERSPALTE=1
[EA3] 1610 FENSTERBREITE=80
[LG4] 1620 FENSTERLAENGE=MAXIMAL+3
[9Y1] 1630 GOSUB 770
[UV1] 1640 RETURN
[PR0] 1650 :
[RN6] 1660 REM * Berechnung der Menü-Parameter *
[YR0] 1670 :
[A32] 1680 MAXIMAL=0
[YT1] 1690 MENUE=0
[R11] 1700 I=1
[GF3] 1710 WHILE I<LEN(KOPF$)
[GN6] 1720 IF MID$(KOPF$,I,1)=" " THEN I=I+1:GOTO 172
0
[PW2] 1730 MENUE=MENUE+1
[584] 1740 STARTSPALTE(MENUE)=I
[H07] 1750 IF MID$(KOPF$,I,1)<>" " THEN I=I+1:GOTO 17
50
[VC7] 1760 NAMENLAENGE(MENUE)=I-STARTSPALTE(MENUE)
[2A1] 1770 WEND
[LR2] 1780 ANZAHL=MENUE
[V21] 1790 I=1
[274] 1800 FOR MENUE=1 TO ANZAHL
[GE4] 1810 STARTSTRING(MENUE)=I+1
[YB5] 1820 BREITE(MENUE)=LEN(MENUES(I+1))
[1D1] 1830 I=I+1
[R55] 1840 IF MENUES(I)<>" " THEN GOTO 1830
[GF6] 1850 LAENGE(MENUE)=I-STARTSTRING(MENUE)
[H69] 1860 IF LAENGE(MENUE)>MAXIMAL THEN MAXIMAL=LAEN
GE(MENUE)
[WE2] 1870 NEXT MENUE
[GW1] 1880 RETURN
[CS0] 1890 :
[6D4] 1900 REM * Pull-Down-Menüs *
[FR0] 1910 :
[1A9] 1920 KOPF$ ="Basic-Kurs Grafik-Befehle DR
AW-Kommandos "
[782] 1930 MENUES(1)=""
[WR4] 1940 MENUES(2)="Information "
[V04] 1950 MENUES(3)=""
[EX3] 1960 MENUES(4)="Ende "
[282] 1970 MENUES(5)=""
[264] 1980 MENUES(6)="CIRCLE "
[RV3] 1990 MENUES(7)="CLS "
[N14] 2000 MENUES(8)="COLOR "
[TX3] 2010 MENUES(9)="DRAW "
[0V3] 2020 MENUES(10)="GET "
[AY3] 2030 MENUES(11)="LINE "
[634] 2040 MENUES(12)="PAINT "
[LA4] 2050 MENUES(13)="PALETTE "
[ZX4] 2060 MENUES(14)="PALETTE USING "
[B04] 2070 MENUES(15)="PMAP "
[U54] 2080 MENUES(16)="POINT "
```

```
[PB4] 2090 MENUES(17)="PRESET "
[Z04] 2100 MENUES(18)="PSET "
[NY3] 2110 MENUES(19)="PUT "
[G74] 2120 MENUES(20)="SCREEN "
[804] 2130 MENUES(21)="VIEW "
[YA4] 2140 MENUES(22)="WINDOW "
[X92] 2150 MENUES(23)=""
[MC4] 2160 MENUES(24)="U anzahl "
[8B4] 2170 MENUES(25)="D anzahl "
[7C4] 2180 MENUES(26)="L anzahl "
[0D4] 2190 MENUES(27)="R anzahl "
[VA4] 2200 MENUES(28)="E anzahl "
[5B4] 2210 MENUES(29)="F anzahl "
[LA4] 2220 MENUES(30)="G anzahl "
[XA4] 2230 MENUES(31)="H anzahl "
[R25] 2240 MENUES(32)="M x-wert,y-wert "
[3B4] 2250 MENUES(33)="A winkel "
[5M4] 2260 MENUES(34)="TA winkel "
[674] 2270 MENUES(35)="C farbe "
[D35] 2280 MENUES(36)="S skalenfaktor "
[W35] 2290 MENUES(37)="X zeichenkette$ "
[045] 2300 MENUES(38)="P farbe,randfarbe"
[7A2] 2310 MENUES(39)=""
[BV1] 2320 RETURN
[7R0] 2330 :
[5W2] 2340 REM * Menü 1 *
[FR0] 2350 :
[LR0] 2360 :
[BK3] 2370 REM * Information *
[VR0] 2380 :
[P28] 2390 PRINT TAB(20)"Menügesteuerte Demonstration
[WY5] 2400 PRINT TAB(20)"der GW-Grafik-Befehle
[DK1] 2410 PRINT
[NB5] 2420 PRINT TAB(20)"von C.Q.Spitzner
[KV1] 2430 RETURN
[PV1] 2440 RETURN
[KR0] 2450 :
[K82] 2460 REM * ENDE *
[UR0] 2470 :
[V41] 2480 END
[2S0] 2490 :
[XV2] 2500 REM * Menü 2 *
[5R0] 2510 :
[2DB] 2520 PRINT TAB(20)"Syntax: CIRCLE (x,y),r,Farbe,
Start,Stop,Verhältnis
[SK1] 2530 PRINT
[2CA] 2540 PRINT TAB(20)"Es wird eine Ellipse mit dem M
ittelpunkt (x,y)
[V57] 2550 PRINT TAB(20)"und dem Radius r gezeichnet.
[4L1] 2560 PRINT
[L3B] 2570 PRINT TAB(20)"Die Parameter 'Farbe', 'Start'
, 'Stop' und 'Verhältnis'
[PM5] 2580 PRINT TAB(20)"können entfallen.
[9V7] 2590 PRINT TAB(20)"Die Parameter 'Start' und 'Sto
p'
[PM7] 2600 PRINT TAB(20)"werden im Bogenmaß angegeben.
[EAB] 2610 PRINT TAB(20)"'Verhältnis' gibt das Verhältn
is der Hauptachsen
[3R3] 2620 PRINT TAB(20)"wieder.
[SV1] 2630 RETURN
[6S4] 2640 PRINT TAB(20)"Syntax: CLS"
[3L1] 2650 PRINT
[3S8] 2660 PRINT TAB(20)"CLS löscht den aktuellen Bilds
chirm"
[8W1] 2670 RETURN
[HW8] 2680 PRINT TAB(20)"Syntax: COLOR Hintergrund,Far
bskala
[LL1] 2690 PRINT
[C98] 2700 PRINT TAB(20)"Dieser Befehl setzt die Farben
für "
[6B6] 2710 PRINT TAB(20)"die Grafik-Modi
[2V4] 2720 PRINT
[KS9] 2730 PRINT TAB(20)"'Hintergrund' darf zwischen 0
und 15 liegen.
[H79] 2740 PRINT TAB(20)"'Farbskala' ist ein Wert zwisc
he 0 und 255.
[3W1] 2750 RETURN
[P37] 2760 PRINT TAB(20)"Syntax: DRAW Zeichenkette"
[EL1] 2770 PRINT
[9F9] 2780 PRINT TAB(20)"DRAW zeichnet ein, mittels 'Ze
ichenkette'
[WR7] 2790 PRINT TAB(20)"vordefiniertes, Grafik-Objekt.
[NK1] 2800 PRINT
[1T9] 2810 PRINT TAB(20)"Der Parameter 'Zeichenkette' b
esteht aus "
[5U9] 2820 PRINT TAB(20)"speziellen Grafik-Befehlen, de
ren Bedeutung
[NZ7] 2830 PRINT TAB(20)"aus Menü 3 ersehen werden kann
[2W1] 2840 RETURN
[C29] 2850 PRINT TAB(20)"Syntax: GET (x1,y1)-(x2,y2),
Feldvariable
[DL1] 2860 PRINT
[269] 2870 PRINT TAB(20)"Dieser Befehl liest einen rech
teckigen "
```

Listing 1. »grafik1« vermittelt Pulldown-Menüs und einen Grafikkurs mit GW-Basic


```
[D09] 2880 PRINT TAB(20)"Grafikbereich in eine Feldvari-
able ein.
[TL1] 2890 PRINT
[0U8] 2900 PRINT TAB(20)"(x1,y1) ist die linke obere Ko-
ordinate.
[G59] 2910 PRINT TAB(20)"(x2,y2) ist die rechte untere
Koordinate.
[UR9] 2920 PRINT TAB(20)"Die Anzahl der benötigten Byte
s kann aus
[N49] 2930 PRINT TAB(20)"folgender Formel berechnet wer-
den:
[8L1] 2940 PRINT
[LS8] 2950 PRINT TAB(20)"Anzahl=INT((x*Wertigkeit+7)/8)
*y + 4
[GL1] 2960 PRINT
[X27] 2970 PRINT TAB(20)"Wertigkeit:

[0M7] 2980 PRINT TAB(20)"             hohe Auflösung: 1
[YD8] 2990 PRINT TAB(20)"             mittlere Auflösung
: 2
[WU1] 3000 RETURN
[6U9] 3010 PRINT TAB(20)"Syntax: LINE (x1,y1)-(x2,y2),
Farbe,B(F),Maske
[7K1] 3020 PRINT
[A2A] 3030 PRINT TAB(20)"Dieser Befehl zeichnet wahlwei-
se eine Linie"
[799] 3040 PRINT TAB(20)"oder ein Viereck auf den Grafi-
k-Schirm."
[LK1] 3050 PRINT
[D48] 3060 PRINT TAB(20)"(x1,y1) ist die Anfangskordin-
ate.
[F59] 3070 PRINT TAB(20)"(x2,y2) ist die rechte untere
Koordinate.
[T0A] 3080 PRINT TAB(20)"Der Parameter 'Farbe' kann je-
nach Auflösung
[S19] 3090 PRINT TAB(20)"einen Wert zwischen 0 und 3 an-
nehmen.
[M9A] 3100 PRINT TAB(20)"Durch Angabe des Parameters 'B
'wird ein Viereck
[HKA] 3110 PRINT TAB(20)"gezeichnet. Durch 'F' wird das
Viereck ausgemalt."
[3A9] 3120 PRINT TAB(20)"'Maske' enthält das Bit-Muster
der Linie
[BV1] 3130 RETURN
[2SB] 3140 PRINT TAB(20)"Syntax: PAINT (x,y),Füllfarbe
,Grenzfarbe,Hintergrund
[PK1] 3150 PRINT
[NBA] 3160 PRINT TAB(20)"Ein Bildschirmbereich wird mit
einer Farbe oder
[D89] 3170 PRINT TAB(20)"in einem bestimmten Muster aus
gefüllt."
[2L1] 3180 PRINT
[8UB] 3190 PRINT TAB(20)"Der Parameter 'Füllfarbe' enhä-
lt Farbe und Füllmuster.
[A8A] 3200 PRINT TAB(20)"der zu füllende Bereich muß mi-
t der Grenzfarbe
[BSA] 3210 PRINT TAB(20)"umrandet sein. 'Hintergrund' d-
efiniert die Hinter-
[WB7] 3220 PRINT TAB(20)"grund-Attribute

[EV1] 3230 RETURN
[ZL8] 3240 PRINT TAB(20)"Syntax: PALETTE Skalenfarbe,
Farbe
[TK1] 3250 PRINT
[Y99] 3260 PRINT TAB(20)"PALETTE änderte eine Farbe der
aktuellen"
[RG6] 3270 PRINT TAB(20)"Farbskala.

[5L1] 3280 PRINT
[YA9] 3290 PRINT TAB(20)"'Skalenfarbe' darf zwischen 0
und 3 liegen"
[7C8] 3300 PRINT TAB(20)"und selektiert eine Farbe der
Skala."
[468] 3310 PRINT TAB(20)"'Farbe' gibt die neue Farbe wi-
eder."
[DV1] 3320 RETURN
[EN8] 3330 PRINT TAB(20)"Syntax: PALETTE USING Integer
-Feld
[SK1] 3340 PRINT
[3F9] 3350 PRINT TAB(20)"PALETTE USING ändert alle Farb-
en einer
[PG6] 3360 PRINT TAB(20)"Farbskala.

[4L1] 3370 PRINT
[W7A] 3380 PRINT TAB(20)"'Integer-Feld' ist das erste v-
on 4 aufeinander"
[00B] 3390 PRINT TAB(20)"folgenden Feldern, die die 4 n-
euen Farben enthalten.
[8V1] 3400 RETURN
[956] 3410 PRINT TAB(20)"Syntax: v = PMAP(x,n)
[LK1] 3420 PRINT
[7MA] 3430 PRINT TAB(20)"PMAP dient zur Umrechnung von
physikalischen
[66A] 3440 PRINT TAB(20)"in anwenderspezifische (Window
) Koordinaten.
```

```
[ZK1] 3450 PRINT
[08A] 3460 PRINT TAB(20)"'x' ist die Koordinate des Umr-
echnungspunktes."
[Z48] 3470 PRINT TAB(20)"'n' ist der Umrechnungsmodus (
0-3)."
[XNA] 3480 PRINT TAB(20)" 0: anwenderspezifisches x -
> physikalisches x"
[CPA] 3490 PRINT TAB(20)" 1: anwenderspezifisches y -
> physikalisches y"
[CPA] 3500 PRINT TAB(20)" 2: physikalisches x -> anwe-
nderspezifisches x"
[TPA] 3510 PRINT TAB(20)" 3: physikalisches y -> anwe-
nderspezifisches y"
[LV1] 3520 RETURN
[X39] 3530 PRINT TAB(20)"Syntax: v = POINT(x,n) oder v
= POINT(n)"
[VK1] 3540 PRINT
[HEA] 3550 PRINT TAB(20)"POINT übergibt Farbe eines Bil-
dpunktes oder
[D9A] 3560 PRINT TAB(20)"die Koordinaten des aktuellen
Grafikpunktes.
[AL1] 3570 PRINT
[6G9] 3580 PRINT TAB(20)"'x' und 'y' enthalten die Koor-
dinaten des
[XW6] 3590 PRINT TAB(20)"gewünschten Bidpunktes.
[FFA] 3600 PRINT TAB(20)"'n' spezifiziert die gewünscht-
e Koordinate:
[0X9] 3610 PRINT TAB(20)" 0 liefert die aktuelle Bild-
Koordinate x
[9X9] 3620 PRINT TAB(20)" 1 liefert die aktuelle Bild-
Koordinate y
[HBA] 3630 PRINT TAB(20)" 2 liefert die aktuelle Windo-
w-Koordinate x
[WBA] 3640 PRINT TAB(20)" 3 liefert die aktuelle Windo-
w-Koordinate y
[2W1] 3650 RETURN
[R68] 3660 PRINT TAB(20)"Syntax: PRESET (x,y), Farbe

[DL1] 3670 PRINT
[YW9] 3680 PRINT TAB(20)"PRESET setzt an der durch 'x,y
' angegebenen
[DP8] 3690 PRINT TAB(20)"Stelle einen Grafikpunkt.

[MK1] 3700 PRINT
[DLA] 3710 PRINT TAB(20)"Die Farbe des Grafikpunktes ka-
nn mittels 'Farbe'"
[TY5] 3720 PRINT TAB(20)"eingestellt werden.
[XV1] 3730 RETURN
[0W7] 3740 PRINT TAB(20)"Syntax: PSET (x,y), Farbe

[8L1] 3750 PRINT
[2N9] 3760 PRINT TAB(20)"PSET setzt an der durch 'x,y
' angegebenen
[8P8] 3770 PRINT TAB(20)"Stelle einen Grafikpunkt.

[ML1] 3780 PRINT
[DMA] 3790 PRINT TAB(20)"Die Farbe des Grafikpunktes ka-
nn mittels 'Farbe'"
[MY5] 3800 PRINT TAB(20)"eingestellt werden.
[SV1] 3810 RETURN
[BW8] 3820 PRINT TAB(20)"Syntax: PUT (x,y), Feldvariab-
le,Modus"
[3L1] 3830 PRINT
[C0A] 3840 PRINT TAB(20)"Dieser Befehl gibt die Grafik-
Informationen,"
[TT9] 3850 PRINT TAB(20)"die mittels GET eingelesen wur-
de wieder.
[FL1] 3860 PRINT
[RBA] 3870 PRINT TAB(20)"'Modus' enthält den Wiedergabe-
modus des Shapes:
[LD6] 3880 PRINT TAB(20)"PSET : Originalmodus
[986] 3890 PRINT TAB(20)"PRESETG: Invertiert
[PBA] 3900 PRINT TAB(20)"AND : Und-Verknüpfung mit d-
em Hintergrund"
[X8A] 3910 PRINT TAB(20)"OR : ODER-Verknüpfung mit
dem Hintergrund"
[USB] 3920 PRINT TAB(20)"XOR : Exklusiv-ODER-Verknüp-
fung mit dem Hintergrund"
[3W1] 3930 RETURN
[H3B] 3940 PRINT TAB(20)"Syntax: SCREEN Modus,Farbe,ak-
t. Seite, sichtb. Seite"
[EL1] 3950 PRINT
[VK9] 3960 PRINT TAB(20)"Dieser Befehl setzt alle wicht-
igen Bild-
[S18] 3970 PRINT TAB(20)"schirmfunktionen.

[UL1] 3980 PRINT
[VKA] 3990 PRINT TAB(20)"'Modus' ist für den Grafikmodu-
s verantwortlich:"
[LF5] 4000 PRINT TAB(20)"0 = Textmodus
[WR7] 4010 PRINT TAB(20)"1 = mittlere Grafikauflösung"
[887] 4020 PRINT TAB(20)"2 = hohe Grafikauflösung
[VK9] 4030 PRINT TAB(20)"Die Parameter 'akt. Seite' und
'sichtb. Seite'"
[YF8] 4040 PRINT TAB(20)"sind nur im Textmodus zugelass-
en."
```



```
[KV1] 4050 RETURN
[F1B] 4060 PRINT TAB(20)"Syntax: VIEW (SCREEN) (x1,y1)
      -(x2,y2),Farbe,Randfarbe"
[XK1] 4070 PRINT
[SEA] 4080 PRINT TAB(20)"VIEW definiert einen Grafik-Bereich als Window."
[RL9] 4090 PRINT TAB(20)"Dieser Bereich kann mit Farbe gefüllt oder"
[AF5] 4100 PRINT TAB(20)"umrandet werden."
[8K1] 4110 PRINT
[HYA] 4120 PRINT TAB(20)"'x1,y1' sind die Koordinaten der linken oberen Ecke"
[LGB] 4130 PRINT TAB(20)"'x2,y2' sind die Koordinaten der rechten unteren Ecke"
[HV1] 4140 RETURN
[3V8] 4150 PRINT TAB(20)"Syntax: WINDOW (SCREEN) (x1,y1)-(x2,y2)"
[WK1] 4160 PRINT
[KR8] 4170 PRINT TAB(20)"Die WINDOW-Anweisung legt fest, welche"
[NH9] 4180 PRINT TAB(20)"Anwenderdaten in ein Bildschirmfenster"
[3Y3] 4190 PRINT TAB(20)"passen."
[7K1] 4200 PRINT
[ØH8] 4210 PRINT TAB(20)"'x1,y1' sind minimalen X- und Y-Werte"
[TH8] 4220 PRINT TAB(20)"'x2,y2' sind maximalen X- und Y-Werte"
[GV1] 4230 RETURN
[CRØ] 4240 :
[EW2] 4250 REM * Menü 3 *
[MRØ] 4260 :
[D42] 4270 GOSUB 5220
[VU8] 4280 PRINT TAB(20)"'U anzahl' bewegt den Grafik-Cursor um"
[LM6] 4290 PRINT TAB(20)"'anzahl' Pixel nach oben"
[7V1] 4300 RETURN
[S32] 4310 GOSUB 5220
[4T8] 4320 PRINT TAB(20)"'D anzahl' bewegt den Grafik-Cursor um"
```

```
[NS6] 4330 PRINT TAB(20)"'anzahl' Pixel nach unten"
[RV1] 4340 RETURN
[842] 4350 GOSUB 5220
[4U8] 4360 PRINT TAB(20)"'L anzahl' bewegt den Grafik-Cursor um"
[XS6] 4370 PRINT TAB(20)"'anzahl' Pixel nach links"
[7W1] 4380 RETURN
[S42] 4390 GOSUB 5220
[VT8] 4400 PRINT TAB(20)"'R anzahl' bewegt den Grafik-Cursor um"
[2X6] 4410 PRINT TAB(20)"'anzahl' Pixel nach rechts"
[KV1] 4420 RETURN
[342] 4430 GOSUB 5220
[HT8] 4440 PRINT TAB(20)"'E anzahl' bewegt den Grafik-Cursor um"
[6T7] 4450 PRINT TAB(20)"'anzahl' Pixel nach rechts oben"
[2W1] 4460 RETURN
[L42] 4470 GOSUB 5220
[3U8] 4480 PRINT TAB(20)"'F anzahl' bewegt den Grafik-Cursor um"
[F18] 4490 PRINT TAB(20)"'anzahl' Pixel nach rechts unten"
[DV1] 4500 RETURN
[Y32] 4510 GOSUB 5220
[GT8] 4520 PRINT TAB(20)"'G anzahl' bewegt den Grafik-Cursor um"
[PW7] 4530 PRINT TAB(20)"'anzahl' Pixel nach links unten"
[XV1] 4540 RETURN
[E42] 4550 GOSUB 5220
[2U8] 4560 PRINT TAB(20)"'H anzahl' bewegt den Grafik-Cursor um"
[VK7] 4570 PRINT TAB(20)"'anzahl' Pixel nach links oben"
[DW1] 4580 RETURN
[Y42] 4590 GOSUB 5220
[PP8] 4600 PRINT TAB(20)"'M x-wert,y-wert' bewegt den Grafik-"
[FL8] 4610 PRINT TAB(20)"Cursor zum Punkt (x-wert, y-wert)."
```

Listing 1. »grafik1« vermittelt Pulldown-Menüs und einen Grafikkurs mit GW-Basic

★ **MultiScreen ist keine Hexerei:**

★ DIE NEUE ★ BILDSCHIRM- ★ DIMENSION

★ für Geschäftsgrafiken,
★ Zeichenprogramme und
★ Spielprogramme aller Art.

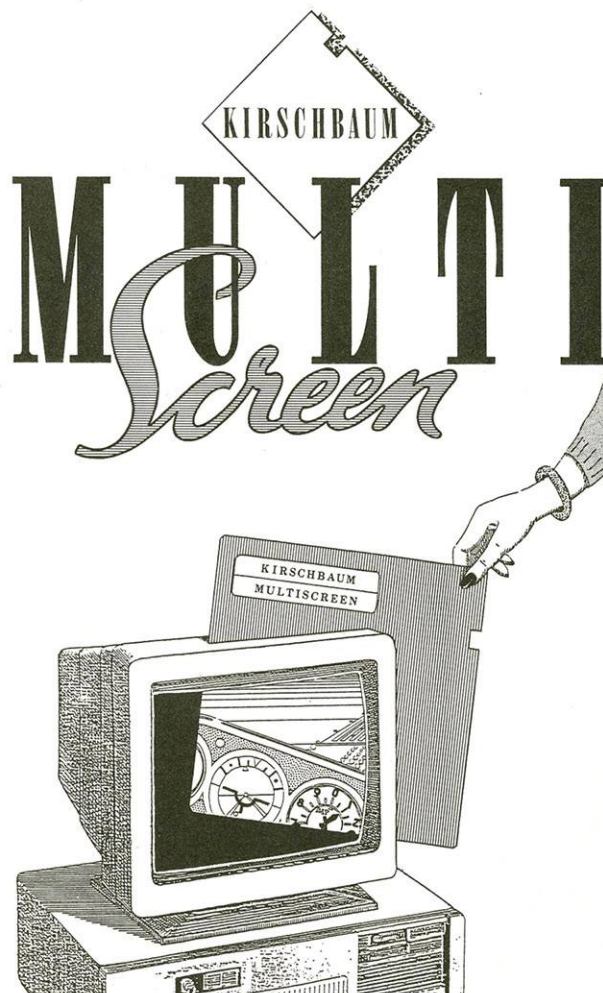
★ Sie kennen das Problem:

★ Auf Ihrem PC mit Monochrom-Monitor und Herculesgrafikkarte funktionieren einfach keine Grafikspiele. Dieselben Schwierigkeiten erleben Sie bei Geschäftsgrafiken, Zeichenprogrammen und CAD-Anwendungen. Der Bildschirm bleibt dunkel.

★ Der Grund: Die meisten Grafikprogramme unterstützen nicht den Hercules-Standard. Selbst das auch beinahe in allen PC's verfügbare GW-BASIC zaubert keine Grafik auf Ihren Bildschirm. Ist es Ihnen nicht auch schon so ergangen? Sie geben immer wieder die Grafikbefehle LINE, CIRCLE, DRAW, etc. ein und sind dabei fast verzweifelt...

★ Schluß damit! Kein dunkler Bildschirm mehr! Wir haben das Problem gelöst!

★ MultiScreen II Version 3.3 DM 112,-
★ (für IBM und Kompatible)
★ MultiScreen 1640 Version 2.2 DM 98,-
★ (nur für AMSTRAD PC 1640 MD)
★ MultiScreen EURO Version 1.1 DM 98,-
★ (nur für Schneider EURO-PC)
★ (unverbindliche Preisempfehlung)



... und das bietet Ihnen MultiScreen:

■ MultiScreen emuliert auf Ihrem IBM XT/AT (und alle Kompatiblen wie z.B. Commodore PC III oder TANDON) mit HERCULES-Monochrom Grafikkarte (oder kompatibler Grafikkarte) folgende Grafikauflösungen:

320 × 200 Punkte 3 Graustufen CGA kompatibel
640 × 200 Punkte monochrom CGA kompatibel
640 × 400 Punkte monochrom Olivetti kompatibel
Herculesgrafik mit 720 × 348 Punkte ohne Einschränkung.

■ Alle BIOS-Aufrufe werden für diese Modi unterstützt.

■ CGA-Programme werden ohne Geschwindigkeitsverlust emuliert.

■ Spielprogramme werden Wirklichkeit: DEFENDER OF THE CROWN, GATO, PITSTOP II, PC-GOLF, SARGON III, GUNSHIP, KING'S QUEST, DIGDUG, BUSHIDO, JET, NEWSROOM, KEYDRAW, PC-PG, THE DRAFTSMAN, PACMAN, STRIKER, LODERUNNER, etc...

■ Verziehen Sie nicht länger auf Grafik bei Programmen wie GWBASIC, TURBO PASCAL, PRINTMASTER, GRAFIK ASSISTANT, etc.

■ Einfache Handhabung bei selbststartenden Programmen. Dadurch keine Probleme bei Spielen wie TAPPER, SUMMER GAMES, WINTER GAMES, CALIFORNIA GAMES, WORLD GAMES, DECATHLON, NIGHTMISSION PINBALL, FLIGHTSIMULATOR von Microsoft.

■ Alle Programme, die bei der Umschaltung von Text auf Grafik und umgekehrt über das BIOS laufen (GWBASIC, BASICA, TURBO PASCAL, Printmaster, PC-PAINT, Framework, Enable, usw.) erscheinen absolut ohne Probleme.

■ Für Programme, bei denen die Umschaltung von Text auf Grafik nicht über das BIOS geht, bietet MultiScreen eine Hot-Key-Taste. Dadurch ist selbst der Flugsimulator ohne Einschränkung lauffähig.

■ Software ohne Probleme auf Festplatte installierbar.

■ MultiScreen ist voll in Assembler geschrieben und bleibt resident im RAM-Speicher, verbraucht jedoch nur 9 KB.

■ Auslieferung mit ausführlicher deutscher Benutzeranleitung.

KIRSCHBAUM SOFTWARE GmbH

Kronau 15, D-8091 Emmering,

Tel. 08067/1220 oder 1051,

Fax. 08067/1053

Schweiz: DSE Datensystems-Engineering AG, Badener Str. 262, CH 8004 Zürich.

Österreich: Überreuter Media Handels- u. Verlagsges. m.b.H., Laudongasse 29, A-1082 Wien.

Wir suchen noch weitere Händler!


```
[SV1] 4620 RETURN
[942] 4630 GOSUB 5220
[478] 4640 PRINT TAB(20) "'A winkel' setzt einen rechten
Winkel"
[SE6] 4650 PRINT TAB(20) "von 0, 90, 180 oder 270 Grad "
[LB5] 4660 PRINT TAB(20) " 0 Grad: winkel=0"
[BD5] 4670 PRINT TAB(20) " 90 Grad: winkel=1"
[OE5] 4680 PRINT TAB(20) "180 Grad: winkel=2"
[SE5] 4690 PRINT TAB(20) "270 Grad: winkel=3"
[LV1] 4700 RETURN
[442] 4710 GOSUB 5220
[OP8] 4720 PRINT TAB(20) "'TA winkel' dreht um einen Win
kel von"
[B45] 4730 PRINT TAB(20) "-360 bis 360 Grad"
[TS8] 4740 PRINT TAB(20) "Ist der Parameter negativ wird
rechts
[U15] 4750 PRINT TAB(20) "herum gedreht."
[ZS8] 4760 PRINT TAB(20) "Ist der Parameter positiv wird
links
[225] 4770 PRINT TAB(20) "herum gedreht."
[LW1] 4780 RETURN
[452] 4790 GOSUB 5220
[MB8] 4800 PRINT TAB(20) "'C farbe' setzt die Farbe für
den "
[A15] 4810 PRINT TAB(20) "DRAW-Befehl "
[TT9] 4820 PRINT TAB(20) "Der Parameter darf bei mittler
er Auflösung
[SU8] 4830 PRINT TAB(20) "einen Wert zwischen 0 und 3 en
thalten."
[609] 4840 PRINT TAB(20) "Der Parameter darf bei hoher A
uflösung
[UU8] 4850 PRINT TAB(20) "einen Wert zwischen 0 und 1 en
thalten."
[EW1] 4860 RETURN
[Z42] 4870 GOSUB 5220
[GR8] 4880 PRINT TAB(20) "'S skalenfaktor' legt die Läng
e der "
[487] 4890 PRINT TAB(20) "einzelnen Bewegungen fest."
[E97] 4900 PRINT TAB(20) "Dabei gilt: S=skalenfaktor/4"
[377] 4910 PRINT TAB(20) " & 0 < skalenfaktor <
256 "
[1W1] 4920 RETURN
[K42] 4930 GOSUB 5220
[8U8] 4940 PRINT TAB(20) "'X zeichenkette$' führt eine w
eiter"
[3U8] 4950 PRINT TAB(20) "Befehlscode-Zeichenkette aus,
die in
[1P9] 4960 PRINT TAB(20) "Variablen 'zeichenkette$' gesp
eichert ist."
[4T8] 4970 PRINT TAB(20) "Der Aufruf erfolgt aus der akt
uellen "
[FZ4] 4980 PRINT TAB(20) "Zeichenkette."
[XW1] 4990 RETURN
[D32] 5000 GOSUB 5220
[PX8] 5010 PRINT TAB(20) "'P farbe,randfarbe' füllt eine
Fläche,"
[3E9] 5020 PRINT TAB(20) "die mit der 'randfarbe' umrand
et ist durch"
[617] 5030 PRINT TAB(20) "die 'farbe' aus."
[CT9] 5040 PRINT TAB(20) "Beide Parameter dürfen je nach
Auflösung,"
[HP9] 5050 PRINT TAB(20) "Werte zwischen 0 und 1 bzw. 0
und 3 annehmen."
[SV1] 5060 RETURN
[MR0] 5070 :
[Y94] 5080 REM * Auf Taste warten *
[WR0] 5090 :
[6K1] 5100 PRINT
[ZR2] 5110 PRINT TAB(20);
[Y71] 5120 COLOR 0,7
[UZ8] 5130 PRINT "Taste";CHR$(29);CHR$(29);CHR$(29);CHR
$(29);CHR$(29);
[1Y1] 5140 COLOR 7,0
[3R2] 5150 TASTE$=INKEY$
[NW3] 5160 IF TASTE$="" THEN 5150
[PE2] 5170 PRINT " ";
[3W1] 5180 RETURN
[ZR0] 5190 :
[P34] 5200 REM * DRAW-Kommandos *
[2R0] 5210 :
[HK1] 5220 PRINT
[AS2] 5230 PRINT TAB(20);
[KY1] 5240 COLOR 0,7
[OH8] 5250 PRINT "DRAW-Befehl - Grafik-Definitionssprac
he"
[CY1] 5260 COLOR 7,0
[5L1] 5270 PRINT
[6W1] 5280 RETURN
```

**Listing 1. »grafik1« vermittelt Pulldown-Menüs und einen
Grafikkurs mit GW-Basic**

```
[C85] 1 REM Programmname: GRAFIK2.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[FR0] 4 REM
[TD4] 100 REM *****
[S95] 110 REM * CIRCLE-Demo (sp151288) *
[ZD4] 120 REM *****
[FK0] 130 :
[OS1] 140 SCREEN 2
[3H5] 145 'Hohe Auflösung einschalten
[R41] 150 CLS
[L54] 155 'Bildschirm löschen
[G42] 160 LOCATE 5,20
[XT2] 165 'Überschrift
[RG3] 170 PRINT "CIRCLE-Demo"
[R42] 180 LOCATE 6,20
[Z93] 190 PRINT "====="
[8K0] 200 :
[VP2] 210 FOR I=1 TO 1000
[YH4] 215 '1000 Kreise oder Scheiben
[NF3] 220 X =INT(RND*640)
[SZ2] 225 'X-Koordinate
[YE3] 230 Y =INT(RND*200)
[YZ2] 235 'Y-Koordinate
[FZ3] 240 RADIUS =INT(RND*100)
[X42] 245 'Radius
[RN3] 250 FARBE =INT(RND*2)
[OS1] 255 'Farbe
[UW0] 260 :
[Z42] 270 CIRCLE (X,Y),RADIUS,FARBE
[UD3] 275 'Kreis zeichnen
[OX0] 280 :
[DA4] 290 IF I/2 =INT(I/2) THEN 310
[M15] 295 'jeden 2. Kreis ausfüllen
[EL4] 300 PAINT (X,Y),FARBE,FARBE
[LG1] 310 NEXT I
[WR1] 320 SCREEN 0
```

**Listing 2. Eine Demonstration des Circle- und Paint-Befehls
durch »grafik2«**

```
[G85] 1 REM Programmname: GRAFIK3.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[FR0] 4 REM
[TD4] 100 REM *****
[LN5] 110 REM * Kuchendiagramm (sp151288) *
[ZD4] 120 REM *****
[FK0] 130 :
[OS1] 140 SCREEN 2
[PD3] 145 'Hohe Auflösung
[R41] 150 CLS
[L54] 155 'Bildschirm löschen
[TK0] 160 :
[V86] 160 LOCATE 5,20 : PRINT "Kuchendiagramm"
[XT5] 170 LOCATE 6,20 : PRINT "====="
[UC1] 180 PRINT
[GR5] 190 PRINT " Bitte 10 Werte eingeben:"
[4C1] 200 PRINT
[BK0] 210 :
[4U5] 220 REM * Zehn Zahlenwerte eingeben *
[HK0] 230 :
[VN2] 240 FOR I = 1 TO 10
[FX5] 250 PRINT " Bitte Wert";I;"eingeben";
[T83] 260 INPUT KUCHEN (I)
[VZ3] 270 MAX = MAX + KUCHEN (I)
[XC4] 275 'Maximalwert berechnen
[6H1] 280 NEXT I
[3L0] 290 :
[GK6] 300 REM * %-Werte berechnen und anzeigen *
[DK0] 310 :
[K15] 320 LOCATE 8,35 : PRINT "PROZENT"
[AR4] 330 LOCATE 9,35 : PRINT "====="
[RK0] 331 :
[XN2] 340 FOR I = 1 TO 10
[LD7] 350 LOCATE 9+I,35 : PRINT KUCHEN (I) / MAX * 10
0 ; "%"
[B23] 355 'Prozentangabe
[SK6] 360 WINKEL (I) = KUCHEN (I) / MAX * 2 * 3.14159
3
[7D4] 365 'Bogenmaß berechnen
[5H1] 370 NEXT I
[2L0] 380 :
[UX5] 390 REM * Kuchendiagramm darstellen *
[CK0] 400 :
[PX2] 410 ANFANG = -6.283186
[F03] 415 'Anfang bei -2 PI
[LK0] 411 :
[WU4] 420 LOCATE 2,60 : PRINT "Grafik"
[WL4] 430 LOCATE 3,60 : PRINT "====="
[TK0] 431 :
[ZN2] 440 FOR I = 1 TO 10
```



```
[3L4] 450 ENDE = ANFANG + WINKEL (I)
[UM3] 455 'Endwert berechnen
[BE5] 460 CIRCLE (500,100),120,,ENDE,ANFANG
[LH3] 465 'Segment zeichnen
[WF2] 470 ANFANG=ENDE
[TV4] 475 'Neuer Anfang = altes Ende
[AH1] 480 NEXT I
[8L0] 481 :
[P42] 490 LOCATE 20,1
```

Listing 3. »grafik3« erzeugt ein Kuchendiagramm

```
[M85] 1 REM Programmname: GRAFIK4.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[FR0] 4 REM
[LT4] 100 REM *****
[926] 110 REM * LINE- & PAINT-Demo (sp151288) *
[TT4] 120 REM *****
[FK0] 130 :
[0S1] 140 SCREEN 2
[3H5] 145 'Hohe Auflösung einschalten
[R41] 150 CLS
[L54] 155 'Bildschirm löschen
[NU1] 160 KEY OFF
[CF5] 165 'Funktionstasten ausblenden
[VK0] 170 :
[125] 180 REM * Füllmuster einlesen *
[1L0] 190 :
[LC2] 200 FOR I=1 TO 25
[PM3] 210 READ FUELLMUSTER
[3V5] 220 FUELL$=FUELL$+CHR$(FUELLMUSTER)
[RG1] 230 NEXT I
[MK0] 240 :
[K42] 250 LOCATE 5,30
[Z43] 255 'Titel schreiben
[XB4] 260 PRINT "LINE- & PAINT-Demo"
[T42] 270 LOCATE 6,30
[994] 280 PRINT "===== "
[3L0] 290 :
[DD6] 300 REM * gefüllte Kästchen zeichnen *
[DK0] 310 :
[WU1] 320 PSET (0,0)
[CZ0] 330 I=1
[P62] 340 WHILE I<200
[WH1] 350 I=I+1
[S63] 360 LINE-(I*3,I),1,BF
[PR2] 365 'Block & Fill
[471] 370 WEND
[2L0] 380 :
[8K3] 390 LINE (0,0)-(639,199),1,B
[R63] 395 'Rahmen zeichnen
[CK0] 400 :
[133] 410 CIRCLE (300,40),80,1
[BH4] 415 'Kreis um Titel zeichnen
[KK0] 420 :
[HK3] 430 PAINT (200,1),FUELL$,1
[W66] 435 'Bereich mit Füllmuster füllen
[831] 440 END
[VK0] 450 :
[KP6] 460 REM * Füllmuster besteht aus 25 Byte *
[1L0] 470 :
[045] 480 DATA 0,0,96,80,80,80,80,96,0,24,36,36
[665] 490 DATA 36,36,24,0,14,16,16,12,2,2,28,0,0
```

Listing 4. »grafik4« demonstriert den Line- und Paint-Befehl

```
[S85] 1 REM Programmname: GRAFIK5.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[FR0] 4 REM
[FC4] 100 REM *****
[Z35] 110 REM * DRAW-Demo (sp151288) *
[NC4] 120 REM *****
[FK0] 130 :
[0S1] 140 SCREEN 2
[PD3] 145 'Hohe Auflösung
[R41] 150 CLS
[L54] 155 'Bildschirm löschen
[NU1] 160 KEY OFF
```

```
[CF5] 165 'Funktionstasten ausblenden
[152] 170 LOCATE 9,41
[M63] 180 PRINT "DRAW-Demo"
[C62] 190 LOCATE 10,41
[HX2] 200 PRINT "===== "
[U32] 210 LOCATE 20,1
[EK0] 220 :
[LF3] 230 SCHRITT=2.554769E-04
[7S5] 235 'Schrittweite für SIN-Funktion
[MK0] 240 :
[FR5] 250 REM * positive Sinus-Funktion *
[UK0] 260 :
[E93] 270 WHILE X < 6.2831853#
[LZ3] 275 'Sinus von 0 bis 2 PI
[MW3] 280 SCHRITT=SCHRITT*1.1
[125] 285 'Schrittweite multiplizieren
[VL2] 290 X=X+SCHRITT
[794] 295 'X-Position berechnen
[9Z2] 300 Y=SIN(X)*100+100
[G84] 305 'Y-Position berechnen
[PP2] 310 PSET (X*70,Y)
[MA4] 315 'Grafik-Cursor setzen
[322] 320 GOSUB 510
[V64] 325 'Schriftzug zeichnen
[S61] 330 WEND
[PK0] 340 :
[UV3] 350 SCHRITT=SCHRITT*1.1
[WK0] 360 :
[NN5] 370 REM * negative Sinus-Funktion *
[2L0] 380 :
[D72] 390 WHILE X > 0
[3V3] 395 'Sinus von 2 PI bis 0
[MW3] 400 SCHRITT=SCHRITT/1.1
[VB4] 405 'Schrittweite teilen
[FL2] 410 X=X-SCHRITT
[L84] 415 'X-Position berechnen
[U13] 420 Y=-SIN(X)*100+100
[S84] 425 'Y-Position berechnen
[YP2] 430 PSET (X*70,Y)
[W44] 435 'Grafik-Cursor setzen
[B22] 440 GOSUB 510
[374] 445 'Schriftzug zeichnen
[071] 450 WEND
[YK0] 460 :
[4R1] 470 GOTO 720
[4L0] 480 :
[TR5] 490 REM * Schriftzug "DOS" zeichnen *
[EK0] 500 :
[G25] 510 DRAW "r20f5d30g5120u5r2u30l2u5"
[Z85] 515 'Zeichnet großes D außen
[0G2] 520 XX=POINT(0)+8
[5G2] 530 YY=POINT(1)+5
[7K2] 540 PSET (XX,YY)
[LK3] 550 DRAW "r9f2d26g219u30"
[CV4] 555 'Zeichnet großes D innen
[UM2] 560 XX=POINT(0)+42
[LG2] 570 YY=POINT(1)-5
[LK2] 580 PSET (XX,YY)
[BE4] 590 DRAW "r20f5d30g5120h5u30e5"
[SA5] 595 'Zeichnet großes O außen
[MF2] 600 XX=POINT(0)+4
[1G2] 610 YY=POINT(1)+5
[3K2] 620 PSET (XX,YY)
[TD4] 630 DRAW "r12f3d24g3112h3u24e3"
[9W4] 635 'Zeichnet großes O innen
[2N2] 640 XX=POINT(0)+46
[FG2] 650 YY=POINT(1)-5
[FK2] 660 PSET (XX,YY)
[WHA] 670 DRAW "r20f5d5110h319g3d6f3r17f5d11g5120h5u5r1
0f3r7e3u3h3117h5u13e6"
[GZ3] 675 'Zeichnet großes S
[UM1] 680 RETURN
[BL0] 690 :
[XZ6] 700 REM * Schriftzug "DOS" negativ zeichnen *
[NK0] 710 :
[FS2] 720 PSET (X*90+3,Y)
[ZC5] 730 DRAW "c0r20f5d30g5120u5r2u30l2u5"
[AG2] 740 XX=POINT(0)+8
[FG2] 750 YY=POINT(1)+5
[HK2] 760 PSET (XX,YY)
[3W3] 770 DRAW "c0r9f2d26g219u30"
[4N2] 780 XX=POINT(0)+42
[XG2] 790 YY=POINT(1)-5
[1K2] 800 PSET (XX,YY)
[3M4] 810 DRAW "c0r20f5d30g5120h5u30e5"
[YF2] 820 XX=POINT(0)+4
[BG2] 830 YY=POINT(1)+5
[DK2] 840 PSET (XX,YY)
[DM4] 850 DRAW "c0r12f3d24g3112h3u24e3"
[CN2] 860 XX=POINT(0)+46
[TG2] 870 YY=POINT(1)-5
[TK2] 880 PSET (XX,YY)
[GTA] 890 DRAW "c0r20f5d5110h319g3d6f3r17f5d11g5120h5u5
r10f3r7e3u3h3117h5u13e6"
[631] 900 END
```

Listing 5. Einen Eindruck des Draw-Befehls vermittelt »grafik5«

Sound-Programmierung auf dem PC

Töne fanden auf dem PC bislang nur als Warnsignale oder als Untermauerung für Spiele Verwendung. Es ist jedoch gar nicht so schwer, auch eigene Melodien zu komponieren, wenn Sie wissen, wie der Timer-Baustein 8253 programmiert werden muß, damit er die gewünschten Töne erzeugt.

Zugegeben, die Tonerzeugung gehört nicht gerade zu den Stärken des PC. So ist beispielsweise Mehrkanalton oder die Regelung der Lautstärke aufgrund des technischen Aufbaus der Schaltung zur Tonerzeugung nicht möglich.

Aber trotzdem eignet sich der eingebaute PC-Lautsprecher dazu, einfache Melodien und Geräusche ertönen zu lassen. Viele Programmiersprachen enthalten in ihrem Befehlssatz sogar Kommandos, mit denen sich über den Lautsprecher Töne ausgeben lassen.

In C oder Assembler ist dies aber beispielsweise nicht der Fall, so daß Sie sich selbst Routinen zur Tonausgabe schreiben müssen. Wenn Sie verstanden haben, wie ein Ton prinzipiell erzeugt wird und welche Funktion die PC-Bausteine bei der Tonerzeugung übernehmen, ist diese Aufgabe jedoch leicht zu lösen.

Töne und Geräusche entstehen, indem eine vibrierende Quelle, beispielsweise die Membrane des PC-Lautsprechers, Luftteilchen in Bewegung versetzt. Dehnt sich die Membrane aus, dann wird die umgebende Luft komprimiert, zieht sie sich zusammen, dann reißt der Druckabfall die Luftteilchen auseinander. Die dabei entstehende Bewegung nennt man Schwingung. Eine Schwingung hat die Kollision vieler Luftpartikel zur Folge. Der Zusammenstoß der Luftpartikel setzt sich als Kettenreaktion in alle Richtungen fort. Diese Kettenreaktion macht sich in Form einer Schallwelle bemerkbar.

Die Membrane eines Lautsprechers wird durch elektrische Impulse in Schwingungen versetzt

Die Lautsprechermembrane des PC wird durch elektrische Impulse in Bewegung versetzt. Weil das Funktionsprinzip eines Computers auf dem Binärsystem basiert, arbeitet der PC-Lautsprecher ebenfalls nur mit zwei Zuständen: Spannung und keine Spannung. Das bedeutet wiederum, daß die Membrane aktiv oder ausgeschaltet ist. Die Anzahl der

Schwingungen der Lautsprechermembrane pro Zeiteinheit wird als Frequenz bezeichnet.

Das physikalische Maß für die Frequenz ist die Einheit »Hertz«, die die Anzahl der Schwingungen pro Sekunde angibt. Je mehr Schwingungen die Lautsprechermembrane innerhalb eines bestimmten Zeitraums erzeugt, desto höher ist die dabei entstehende Frequenz.

Die von einer Lautsprechermembrane ausgesandten Schwingungen nennt man Töne. Grundsätzlich unterscheidet man zwischen akustischen und musikalischen Tönen. Während akustische Töne beliebige Frequenzen besitzen können, sind die Frequenzen für die Töne einer musikalischen Oktave genau festgelegt.

Die einzelnen Töne und Halbtöne einer Oktave lassen sich nach einem festen Schema berechnen

Tabelle 1 zeigt die Frequenzen für alle Töne der unteren acht Oktaven. Unterhalb der Oktave, die mit einer Frequenz von 16,35 Hz beginnt, existiert keine weitere Oktave, weil der Mensch erst Frequenzen ab 16 Hz als Töne wahrnimmt. Alle Frequenzen unterhalb dieser Grenze werden nur als Brummen empfunden.

Einigen Frequenzen sind also entsprechende Tonbezeichnungen zugeordnet. Will man beispielsweise ein »eingestrichenes C« ertönen lassen, dann muß der Lautsprecher den Angaben in Tabelle 1 zufolge knapp 262mal in einer Sekunde hin- und herschwingen. Das »eingestrichene C« entspricht damit einem Ton mit einer Frequenz von knapp 262 Hertz.

Der Bezugspunkt für alle Töne in Tabelle 1 ist der Ton A in der eingestrichenen Oktave mit einer Frequenz von 440 Hz. Diesen Ton bezeichnet man auch als Kammerton A. Weil die Frequenz des Tons A in der benachbarten Oktave stets doppelt oder halb so groß ist wie die ursprüngliche Frequenz und eine Oktave aus 12 Halbtönen besteht, ist der Unterschied zwischen zwei benachbarten Halbtö-

nen stets ein Faktor mit dem Wert der zwölften Wurzel aus 2.

Um von der Frequenz eines Halbtons ausgehend die Frequenz des unmittelbar folgenden Halbtons zu berechnen, multiplizieren Sie dazu einfach die Frequenz des ersten Halbtons mit 1,0594631.

Im PC sind mehrere Bausteine für die Tonerzeugung zuständig; besonders wichtig ist der Timer-Baustein 8253. Der 8253 erfüllt im PC viele Aufgaben, zu denen die Ansteuerung des Tonausgangs, die Überwachung der Echtzeituhr, der RAM-Refresh und der Zugriff auf einen Kassettenrekorder gehören.

Im AT arbeitet eine erweiterte Version des 8253, der 8254, der sich aber grundsätzlich nur durch die verbesserte Architektur von seinem Vorgängermodell unterscheidet.

Der Timer-Baustein 8253 enthält drei Kanäle, die die Systemuhr, den RAM-Refresh und den PC-Lautsprecher steuern

Der Timer-Baustein 8253 verwaltet drei Kanäle, die unterschiedliche Aufgabengebiete abdecken. Der Kanal 0 (Portadresse 40hex) steuert die Systemuhr, und Kanal 1 (Portadresse 41hex) wird benutzt, um dem DMA-Controller den Takt vorzugeben, in dem die dynamischen RAMs aufgefrischt werden müssen. Kanal 2 (Portadresse 42hex) ist direkt mit dem Lautsprecherausgang verbunden und aus diesem Grund für die Sound-Programmierung von besonderem Interesse.

Die Hauptaufgabe des Timer-Bausteins besteht im Zählen. Für jeden Kanal ist ein Zählerregister zuständig, dessen Inhalt im Eingangstakt des Bausteins fortlaufend dekrementiert wird. Hat ein Zähler den Wert Null erreicht, dann wird der logische Pegel des Ausgangssignals gewechselt (die eine Hälfte einer Schwingung), und das Zählerregister erhält wieder den Initialisierungswert.

Vielleicht werden Sie sich fragen, warum der PC für diese Aufgabe extra einen Baustein benutzt und das Zählen nicht per Software erledigt. Die Antwort ist einleuchtend: Weil der PC mit Interrupts arbeitet, kann es sein, daß eine softwaregesteuerte Zählroutine unterbrochen wird. Die Synchronisation mit dem Prozessor und der Hardware wäre in diesem Fall fast undurchführbar und ungenau. Ein zusätzlicher Baustein, der diese Timer-Aufgaben übernimmt, ist in diesem Fall eine einfache und elegante Lösung des Problems.

Neben den drei Zählerregistern für die Kanäle des Timer-Bausteins be-

DOS EXTRA 6: Alles über Basic

Jetzt neu!

**EXTRA
EXTRA
EXTRA
EXTRA
EXTRA
EXTRA**

...und vieles mehr!

**Schneller Basic-Compiler
zum Abtippen**

GW-Basic selbst erweitert

**Basic intern: Token, Varia-
blen, Speicherverwaltung
Datenbankprogramm mit
Menüsteuerung**

**Turbo-Basic-Funktions-
Bibliothek für die CGA-Karte
Strukturierte Programmie-
rung in Quick Basic 4.0**

Best-Nr. 383

18,- DM*



Noch über den Verlag erhältlich:

DOS EXTRA 1: Ausverkauft

DOS EXTRA 2: Ausverkauft

DOS EXTRA 3: Für glasklare Informationen

- Der große DOS-Kurs »Der PC – glasklar in überarbeiteter Form
- Datenfernübertragungs-Programm in 100% Assemblercode
- Sprachdigitalisierung mit dem PC
- Hercules-Grafik-Toolbox in Turbo Pascal
- Grafikfunktionen für Turbo C



Best-Nr. 356 18,- DM*

DOS EXTRA 4: Was Sie schon immer interessiert hat

- Desktop Publishing selbst gemacht mit SDTP
- Speicherresidente Programme schreiben, analysieren und entfernen
- Alles über MS-DOS-Interrupt 21 hex
- Grundlagen zu Grafikkarten



Best-Nr. 381 18,- DM*

DOS EXTRA 5: Von Basic bis BIOS

- Aufbau und Programmierung der wichtigsten Grafikkarten
- Basic für Einsteiger und Profis
- Die Interrupts des BIOS im Detail
- Gerätetreiber-Grundlagen



Best-Nr. 382 18,- DM*

Die Ausgaben DOS EXTRA 3, DOS EXTRA 4 und DOS EXTRA 5 sind nur noch über den Verlag zu beziehen. Zu allen Ausgaben ist jedoch die Databox im Format 5 1/4" oder 3 1/2" MS-DOS erhältlich. Bitte benutzen Sie die Bestellkarte.

* Unabhängig von der Anzahl der bestellten Zeitschriften berechnen wir für das Inland 3,- DM bzw. für das Ausland 5,- DM Porto und Verpackung.

DMV Verlag · Postfach 250 · 3440 Eschwege

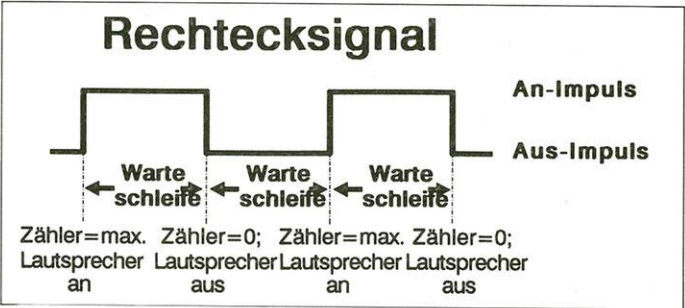


Bild 1. Warteschleifen spielen bei der einfachen Tonausgabe eine wichtige Rolle

stimmt ein Steuerregister mit der Portadresse 43hex die Funktionsweise des 8253. Den Aufbau und die Bedeutung der verschiedenen Belegungen dieses Registers können Sie der Tabelle 2 entnehmen.

Für die Ansteuerung des Lautsprechers wird die Betriebsart angewählt, die ein Rechtecksignal erzeugt. Um den Lautsprecherkanal anzusprechen, legen Bit 6 und 7 des Steuerregisters fest, daß das Zählerregister des Kanal 2 manipuliert werden soll. Gleichzeitig legen Sie über Bit 4 und 5 des Steuerregisters die Anweisung zum Lesen des Zählerwertes fest. Der neue Zählerstand wird dann über die Portadresse 42hex in das Zählerregister des Kanal 2 geschrieben. Beachten Sie dabei, daß Sie zuerst das niederwertige und dann das höherwertige Byte übergeben.

Wenn Sie beispielsweise den Kamerton A über den PC-Lautsprecher ausgeben lassen wollen, müssen Sie

den Timer-Baustein 8253 so programmieren, daß er 440mal in der Sekunde ein elektrisches Signal erzeugt, das die Lautsprechermembrane zum Schwingen anregt. Um dieses Ergebnis zu erreichen, übergeben Sie dem 8253 über die Portadresse 42hex den Wert, der die gewünschte Frequenz festlegt (ein praktisches Beispiel folgt später).

Der Parallel-Portbaustein 8255 arbeitet mit drei, jeweils 8 Bit breiten Peripherieschnittstellen

Der 8253 ist jedoch nicht der einzige Baustein, der im PC für die Tonerzeugung zuständig ist. Fast alle Ein- und Ausgabeoperationen des PC steuert der programmierbare Parallel-Portbaustein 8255. Beim 8255 handelt es sich um einen Baustein, der in verschiedenen Anwendungen als Peri-

perieschnittstelle fungiert. Im PC wird er hauptsächlich für die Verwaltung der Tastatur und zur Kontrolle der Konfigurationsdaten verwendet. Der 8255 verfügt über drei parallele, 8 Bit breite Peripherieschnittstellen (Ports) A, B und C, von denen im PC zwei für Eingabe- und eine für Ausgabezwecke programmiert sind. Die Ausgabeschnittstelle B sprechen Sie über die Portadresse 61hex an, und den beiden Eingabeschnittstellen A und C sind die Portadressen 60hex und 62hex zugewiesen.

Die unteren beiden Bit der Ausgabeschnittstelle (Port B) steuern den Lautsprecher an

Die unteren beiden Bit der Ausgabeschnittstelle steuern den Lautsprecher an. Bit 0 stellt dabei eine Verbindung zum Timer-Baustein 8253 her. Ist Bit 0 gesetzt, dann wird der 8253 veranlaßt, ein Rechtecksignal auf Kanal 2 zu erzeugen. Wenn Bit 1 zusätzlich gesetzt ist, wird der Lautsprecher zur Tonerzeugung freigegeben. Das bedeutet, daß das Rechtecksignal des 8253 an den Lautsprecher weitergeleitet wird und über die schwingende Membrane einen Ton erzeugt. Der Lautsprecher bleibt solange aktiv, bis Bit 0 wieder gelöscht wird. Ist Bit 0 gelöscht, dann generiert der 8253 kein Rechtecksignal.

Neben den drei Registern, die die Werte der drei Peripherieschnittstellen des 8255 zwischenspeichern, enthält der Baustein ein Steuerregister

Die wichtigsten Tonfrequenzen auf einen Blick									
Oktave 1 (Subkontra-Oktave)		Oktave 2 (Kontra-Oktave)		Oktave 3 (große Oktave)		Oktave 4 (kleine Oktave)		Oktave 5 (eingestrichen)	
Oktave 6 (zweigestrichen)		Oktave 7 (dreigestrichen)		Oktave 8 (vieregestrichen)					
Ton	Frequenz	Ton	Frequenz	Ton	Frequenz	Ton	Frequenz	Ton	Frequenz
C	16,35	C	32,70	C	65,41	C	130,81	C	261,63
Cis	17,32	Cis	34,65	Cis	69,30	Cis	138,59	Cis	277,18
D	18,35	D	36,71	D	73,42	D	146,83	D	293,66
Dis	19,45	Dis	38,89	Dis	77,78	Dis	155,56	Dis	311,13
E	20,60	E	41,20	E	82,41	E	164,81	E	329,63
F	21,83	F	43,65	F	87,31	F	174,61	F	349,23
Fis	23,12	Fis	46,25	Fis	92,50	Fis	185,00	Fis	369,99
G	24,50	G	49,00	G	98,00	G	196,00	G	392,00
Gis	25,96	Gis	51,91	Gis	103,83	Gis	207,65	Gis	415,30
A	27,50	A	55,00	A	110,00	A	220,00	A	440,00
Ais	29,14	Ais	58,27	Ais	116,54	Ais	233,08	Ais	466,16
H	30,87	H	61,74	H	123,47	H	246,94	H	493,88

Tabelle 1. Alle Töne der unteren acht Oktaven mit den zugehörigen Frequenzen

zur Verwaltung, das sich über die Portadresse 63hex ansprechen läßt. Das Steuerregister des 8255 legt fest, welche Kanäle für die Ein- und Ausgabe zuständig sind. Weil das Betriebssystem diese Werte jedoch automatisch initialisiert, ist das Steuerregister damit für die Tonerzeugung ohne Bedeutung.

Es existieren grundsätzlich zwei verschiedene Verfahren, um mit dem PC über den Timer-Baustein 8253 und den Parallel-Portbaustein 8255 Töne zu erzeugen.

Das Verfahren zur Tonerzeugung durch direktes Ansteuern des Lautsprechers hat seine Vor- und Nachteile

Die erste Methode, die Listing 1 beispielhaft verdeutlicht, verwendet nur den 8255. Der Prozessor des PC kontrolliert in diesem Fall den Lautsprecher direkt und muß aus diesem Grund die Aufgaben übernehmen, die normalerweise vom Timer-Baustein erledigt werden. Damit ist das Verfahren zwar etwas unübersichtlich, erlaubt aber die direkte Beeinflussung der Tonprogrammierung und macht auf diese Weise auch spezielle Geräuscheffekte oder die Ausgabe von digitalisierten Sounds möglich.

Die Tonerzeugung über den 8255 erfordert nichts weiter, als das mit dem Lautsprecher verbundene Frequenz-Bit 1 der Ausgabeschnittstelle B (Portadresse 61hex) an- und auszuschalten. Wenn ein Programm das Bit allerdings zu schnell setzt und wieder löscht, ist die daraus resultierende Frequenz für das menschliche Gehör zu hoch. Aus diesem Grund müssen, wie in Bild 1 gezeigt, Warteschleifen zwischen das An- und Abschalten des Lautsprechers eingefügt werden.

Ein großer Nachteil dieses Verfahrens zur Tonerzeugung liegt in der Abhängigkeit des Programms vom Hardware-Aufbau eines PC. Je schneller ein PC getaktet ist, desto höher ist der Ton, den er mit Listing 1 erzeugt, und umgekehrt.

Viel komfortabler ist das Verfahren der Tonerzeugung über den Timer-Baustein 8253

Das allergrößte Manko dieser Methode besteht aber in ihrer Anfälligkeit gegenüber Interrupts. Tritt während der Tonerzeugung ein Interrupt auf, so kann es durchaus geschehen, daß der zu erzeugende Ton verzerrt wird oder gar nicht ertönt.

Das zweite Verfahren zur Tonerzeugung wird von diesen Problemfällen

nicht betroffen. In diesem Fall bedienen Sie sich des Timer-Bausteins 8253, damit sowohl Tonlänge und Tonhöhe als auch die Verarbeitungsgeschwindigkeit nicht von der Taktfrequenz des verwendeten Computers abhängen und störende Interrupts die Tonausgabe nicht verzerren. Wie Sie bereits gelesen haben, erzeugt nämlich der Timer-Baustein das Rechtecksignal für den Lautsprecher, ohne die Arbeit des Prozessors zu beeinflussen. Die Tonausgabe über den Timer-Baustein ist in Bild 2 dargestellt.

Der Wert im Zählerregister für Kanal 2 bestimmt die Frequenz des zu erzeugenden Tons

Damit der 8253 einen Ton der gewünschten Frequenz erzeugt, müssen Sie in das Zählerregister für Kanal 2 (Portadresse 42hex) einen 16-Bit-Wert schreiben, durch den die interne Taktfrequenz des 8253 von exakt 1193181,7 Hz geteilt wird. Das Ergebnis dieser Division gibt die Frequenz für das zu erzeugende Rechtecksignal an, das die Lautsprechermembrane zum Schwingen bringt.

Vor der Übergabe des Frequenzteilers an den 8253 müssen Sie jedoch erst das Steuerregister über die Portadresse 43hex mit dem Wert B6hex laden,

Bit-Kombination	Bedeutung
xxxxxxx0	Binärzähler
xxxxxxx1	BCD-Zähler
xxxx000x	Interrupt-Erzeugung
xxxx001x	Programmierbares Monoflop
xxxx010x	Taktgenerator
xxxx011x	Rechtecksignal-Generator
xxxx100x	Triggerung (Auslösung) des Ausgangs durch Software
xxxx101x	Triggerung (Auslösung) des Ausgangs durch Hardware
xxxx110x	Reserviert
xxxx111x	Reserviert
xx00xxxx	Zähler zum Lesen übernehmen
xx01xxxx	Niederwertiges Byte (LSB) für Zähler laden
xx10xxxx	Höherwertiges Byte (MSB) für Zähler laden
xx11xxxx	Erst niederwertiges, dann höherwertiges Byte für Zähler laden
00xxxxxx	Zähler 0 benutzen
01xxxxxx	Zähler 1 benutzen
10xxxxxx	Zähler 2 benutzen
11xxxxxx	Reserviert

Tabelle 2. Über das Steuerregister des Timer-Baustein 8253 wird dessen Betriebsmodus festgelegt

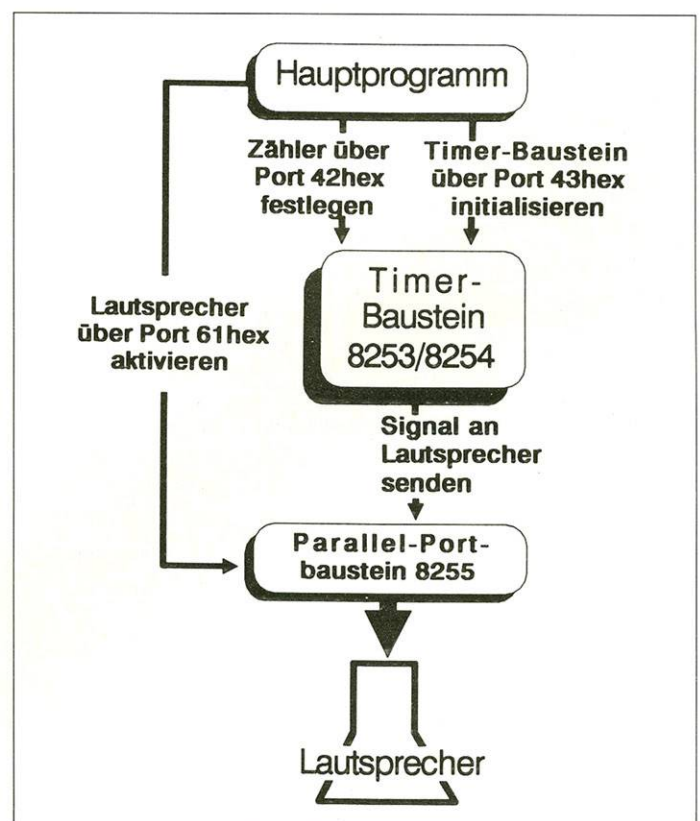


Bild 2. Der Timer-Baustein 8253 des PC kontrolliert die Tonerzeugung

um den Baustein für den Lautsprecherbetrieb zu initialisieren.

Die interne Taktfrequenz des 8253 von 1193181,7 Hz beträgt ein Zwölftel des Systemtaktes von 14318180 MHz, aus dem beispielsweise durch Teilung von 3 auch der typische 4,77-MHz-Prozessortakt eines IBM-PC gewonnen wird. Doch auch in PCs, die mit höheren Taktfrequenzen als 4,77 MHz arbeiten, verwendet der 8253 intern einen Takt von 1193181,7 Hz, so daß dieser Wert für alle PCs, ATs und auch 386er PCs gültig ist.

Die Tonerzeugung über ein Assembler-Programm ist viel einfacher als Sie glauben

Möchten Sie beispielsweise mit dem Timer-Baustein 8253 den Kammerton A erzeugen, so müssen Sie den Wert 2712 in das Zählerregister des Kanal 2 schreiben, weil 1193181,7 geteilt durch 2712 den Frequenzwert 440 ergibt. Sie berechnen also den Frequenzteiler für den 8253, indem Sie die interne Taktfrequenz durch die Frequenz des gewünschten Tons teilen.

Wenn Sie einer Assembleroutine zur Tonerzeugung im DI-Register die Frequenz mitteilen, berechnen die folgenden drei Programmzeilen den korrekten Wert für das Zählerregister im 8253:

```
mov dx,12h
mov ax,34dch
div di
```

In diesem Beispiel wird die interne Taktfrequenz des 8253 als Hexadezimalwert 1234DChex in das Registerpaar DX:AX geschrieben und durch den Frequenzwert in DI geteilt. Das AX-Register enthält das Ergebnis dieser Division.

Eine Zeitschleife garantiert für Töne mit unterschiedlichen Frequenzen einheitliche Längen

Um für Töne verschiedener Frequenzen eine konstante Länge zu garantieren, benötigen Sie in vielen Fällen eine Zeitschleife. Eine Zeitverzögerung von zehn Millisekunden erreichen Sie auf einem PC mit einer Taktfrequenz von 4,77 MHz beispielsweise durch die beiden Assembler-Programmzeilen

```
milli:mov cx,2808
schleif:loop schleif
```

Der Wert 2808 errechnet sich aus der Taktfrequenz des PC und den Ausführungszeiten der Assemblerbefehle. Der Prozessor führt den Mov-Befehl in 4 Taktzyklen aus und die Loop-Funktion entweder in 17 Takten (wenn eine Verzweigung stattfindet) oder in 5 Takten (keine Verzweigung). Weil die Routine durch den Loop-Befehl immer verzweigt, bis das CX-Register den Wert 1 enthält, führt sie bei einem Schleifenzählerwert von 2802

$4 + 17 \times (2808 - 1) + 5 = 47728$

Taktzyklen aus. Die Gesamtzeit der Schleife errechnet sich wiederum aus

der Länge der Taktzyklen. Eine Taktfrequenz von 4,77 MHz (exakt 4772727 Hz) führt zu einem Taktzyklus von

$1s / 4772727 = 209,5 \text{ ns}$

Dauer. Damit ergibt sich eine Gesamtdauer der Schleife von

$47728 \times 209,5 \text{ ns} = 10 \text{ ms}$

Besitzen Sie einen PC, dessen Prozessor mit einer höheren Rate getaktet ist, dann müssen Sie den Multiplikationswert lediglich den Umständen anpassen. Dies geschieht über die Gleichung

$\text{wert} = (n / 10^{-4} \cdot 9) / 17 + 1$

Setzen Sie in diese Formel beispielsweise für n eine Taktfrequenz von 8 MHz ein, so erhalten Sie einen Schleifenzählerwert von 4706.

Listing 2 zeigt Ihnen an einem konkreten Beispiel, wie Sie Töne beziehungsweise vollständige Melodien über den Lautsprecher Ihres PC ausgeben. Das Programm liest jede Note und die zugehörige Tondauer aus den Tabellen »note« und »laenge«, die sich im Datensegment befinden. Der Wert Null kennzeichnet das Ende der Tabellen.

Für die Tonerzeugung verwendet das Programm das zweite beschriebene Verfahren über den Timer-Baustein 8253. Wichtig ist in diesem Zusammenhang, daß Sie alle Register vor der Tonausgabe auf dem Stapel (Stack) ablegen, um so sicherzustellen, daß keine Daten verloren gehen.

(Markus Zietlow/ma)

```
[P30] 1: ;
[7XA] 2: ;Assembler-Programm, das einen Pieps auf dem Lautsprecher erzeugt

[P30] 3: ;
[KN2] 4: tonlaenge equ 800
[UK2] 5: frequenz equ 150
[8B2] 6: port equ 61h
[P30] 7: ;
[X54] 8: code segment para 'code'
[VU2] 9: pieps proc far
[DK2] 10: assume cs:code
[3S3] 11: push ax ;Sichern
[M23] 12: push bx ;der
[LH4] 13: push cx ;wichtigste

n
[GY4] 14: push dx ;Speicherwe
rte
[TG5] 15: cli ;Interrupts
sperrn
[L68] 16: mov dx,tonlaenge ;Länge des T
ons in DX ablegen
[S57] 17: in al,port ;Status von
Ausgabe-Port B
[8P9] 18: and al,1111110b ;holen, kei
ne Rechteckschwingung erzeugen
[9M8] 19: ton: or al,0000010b ;und Lauspr
eher aktivieren
[LH5] 20: out port,al ;Status setz
en
[BX5] 21: mov cx,frequenz ;Warteschlei
fe
[LK6] 22: hochton: loop hochton ;aktivieren
[FM8] 23: and al,11111101b ;Erzeugte Re
chteckschwingung nicht
```

```
[ZL8] 24: out port,al ; an den Lau
tsprecher weiterleiten
[BX5] 25: mov cx,frequenz ;Warteschlei
fe
[7M6] 26: tiefton: loop tiefton ;aktivieren
[RG9] 27: dec dx ;Zähler für
die Tonlänge dekrementieren
[ST6] 28: jnz ton ;Wenn <> 0,
Sprung zu 'ton'
[7V5] 29: sti ;Interrupts
freigeben
[524] 30: pop dx ;Die auf dem
[EL5] 31: pop cx ;Stapel ges
icherten
[PD3] 32: pop bx ;Werte
[SE5] 33: pop ax ;wiederhers
tellen
[M45] 34: mov ah,4ch ;Rückkehr in
die
[SH4] 35: int 21h ;MS-DOS-Ebe
ne
[8A6] 36: pieps endp ;Ende der Pr
ozedur
[517] 37: code ends ;Ende des Co
de-Segments
[TK5] 38: end pieps ;Ende des Pr
ogramms
```

Listing 1. Das Programm »beep.asm« erzeugt auf dem PC einen simplen Piepton


```
[P30] 1: ;
[FRA] 2: ;Assembler-Programm zur Demonstration der Sound-Eigenschaften
[P30] 3: ;
[P44] 4: data segment para 'data'
[VN6] 5: freq dw 98,110,123,131,147,147,1
47,147,165,131
[DR5] 6: dw 196,165,147,165,131,196,1
65,147,147,131
[9K5] 7: dw 131,131,131,123,123,123,1
23,110,123,110
[BL5] 8: dw 98,123,147,147,147,131,1
31,131,131,123
[Z34] 9: dw 123,123,123,110,123,110,
98
[X51] 10: dw 0
[RW6] 11: laenge dw 12 dup (100), 1 dup (400)
, 4 dup (100)
[OT5] 12: dw 1 dup (400), 28 dup (100)
, 1 dup (400)
[DU1] 13: data ends
[P30] 14: ;
[X34] 15: code segment para 'code'
[TV2] 16: proc far
[723] 17: assume cs:code, ds:data
[3S3] 18: push ax ;Sichern
[M23] 19: push bx ;der
[LH4] 20: push cx ;wichtigsten
[GY4] 21: push dx ;Speicherwer
te
[5A5] 22: mov ax,seg data ;Datensegment
[K65] 23: mov ds,ax ;initialisie
ren
[SC5] 24: lea si,freq ;Tabellenzeig
er
[Z05] 25: lea bp,ds:laenge ;einrichten
[5S9] 26: spielen: mov di,[si] ;Folgenden To
n aus Tabelle holen
[137] 27: cmp di,0 ;Wenn die Mel
odie beendet ist,
[RL5] 28: je schluss ; Sprung zum
Ende,
[OE8] 29: mov bx,ds:[bp] ; ansonsten T
ondauer ermitteln
[CY5] 30: mov al,182 ;Ausgaberegis
ter für
[5C7] 31: out 43h,al ; Timer-Baust
ein 8253 schreiben
[T85] 32: mov dx,12h ;Für den zwei
ten
[YY6] 33: mov ax,34dch ; Zeitgeber d
ie benötigte
[YP5] 34: div di ; Frequenz be
stimmen
[H37] 35: out 42h,al ;Das niederwe
rtige und das
[CH8] 36: mov al,ah ; höherwertig
e Byte des errechneten
[KV7] 37: out 42h,al ; Wertes an d
en 8253 übermitteln
[PZ8] 38: in al,61h ;Aktuellen In
halt des Ausgaberegisters
[605] 39: mov ah,al ; des PPI sic
hern
[7T5] 40: or al,3 ; und den Lau
tsprecher
[AR4] 41: out 61h,al ; einschalten
[LD7] 42: mov cx,2808 ;Einen Zeitin
tervall
[CN5] 43: schweif: loop schweif ; warten
[WY6] 44: dec bx ;Zähler dekre
mentieren, bis
[Z05] 45: jnz milli ; er gleich 0
ist
[NV6] 46: mov al,ah ;Ursprünglich
e PPI-Werte
[ZP5] 47: out 61h,al ; wiederherst
ellen
[F69] 48: add si,2 ;Zeiger der T
on- und Lautstärken-Tabelle
[9C5] 49: add bp,2 ; um zwei erh
öhen,
[N08] 50: jnz spielen ; bis Tabelle
nende erreicht ist
[FN5] 51: schluss: pop dx ;Die auf dem
[EL5] 52: pop cx ; Stapel gesi
cherten
[PD3] 53: pop bx ; Werte
[SE5] 54: pop ax ; wiederherst
ellen
[M45] 55: mov ah,4ch ;Rückkehr in
die
[SH4] 56: int 21h ; MS-DOS-Eben
e
[3G6] 57: musik endp ;Ende der Pro
zedur
[U17] 58: code ends ;Ende des Cod
e-Segments
[SM5] 59: end musik ;Ende des Pro
gramms
```

Listing 2. »lied.asm« zeigt, wie wenig Aufwand nötig ist, um Melodien zu spielen

... make it easy!!!

Neu!!!

PC TOOLS Deluxe Version 5 (engl.)

von Central Point Software

2

DOS 4.0 Shell

- + Maus-Unterstützung
- + kompl. Daten-Rekonstruktion
- + Optimierung der Plattenbelegung
- + Caching
- + schnelle, bequeme Datensicherung
- + Taschenrechner
- + Notizbuch und Clipboard
- + Ideen- u. Gedankenordner
- + Terminkalender
- + Dateikomprimierung
- + res. Datenkommunikation
- + "dBASE"-File Editor
- + Makro-Prozessor
- + modernste Fenstertechnik

...Dinge braucht Ihr PC:
DOS und PC TOOLS

DM 159,60 (engl. mit dt. Kurzanleitung)

dt. Komplettversion in Vorbereitung

Update für alle Versionen gegen alte Registrierkarte DM 89,50

Version 4.3 komplett Deutsch DM 128,85

zuzüglich Versandkosten

Preisänderungen vorbehalten.

Weitere US-Software, teilweise mit deutschem Handbuch:

COPYIIPC, OPTION BOARD, 1 DIR Plus, BackEZ, PrintQ, LaserMate, BROOKLYN BRIDGE, EasyFlow, DesqView 2.2, The Documentor, Automenu, InstaPlan, Personal REXX 2.0, KEDIT 4.0, Personal VM, SPF/PC 2.0, FormWorx, SmartNotes, ADD-IN's für LOTUS und SYMPHONY: ALWAYS, @BASE, SeeMore, Look & Link.

Händleranfragen erwünscht.

MS

Werner Kortmann

Kantstraße 4
6239 Eppstein

Tel. 06198-32498
FAX. 06198-1703

Dateiverwaltung mit GW-Basic

Ob es nun Texte oder Artikellisten, Kalkulationsdaten oder Umsatzstatistiken sind: Fast immer dienen Personalcomputer dazu, Daten zu verwalten und auf der Diskette oder der Festplatte zu speichern. Welche Befehle Basic für diesen Zweck bereithält, erfahren Sie in diesem Beitrag.

Das Thema »Dateiverwaltung« gehört zu den wichtigsten Bereichen in der Computertechnik. Schnelle Zugriffsmöglichkeiten auf externe Daten sind dabei ein wichtiges Kriterium für die Leistungsfähigkeit eines Computersystems. Daten, die auf der Diskette oder Festplatte zu speichern sind, werden in Dateien organisiert. Eine Datei ist ein logisch zusammenhängender Block von Daten, dem ein Name zugewiesen wird. Der Datenblock besteht aus einer Aneinanderreihung von Datensätzen. Jeder dieser Datensätze enthält bestimmte Informationen in einer festgelegten Struktur. Das Telefonbuch zum Beispiel ist eine derartige Datei. Hier sieht ein typischer Datensatz wie folgt aus:

Inkognito, Ingo, Inspektor, Ichweißnichtweg 11, 123 456

Ein Datensatz besteht hier also aus fünf Feldern: Jeweils ein Feld für Namen, Vornamen, Beruf, Straße und Telefonnummer.

Komfort und Geschwindigkeit des Zugriffs auf eine Datei hängen im wesentlichen von deren interner Organisation ab. In einer sequentiellen Datei dürfen Daten variabler Länge stehen; sie können aber nur hintereinander gelesen und geschrieben werden. Das Ende eines einzelnen Datensatzes wird dabei durch einen Wagenrücklauf (»CR«: Carriage Return, ASCII-Wert 13) mit nachfolgendem Zeilenvorschub (»LF«: Line Feed, ASCII-Wert 10) gekennzeichnet. Mit einem einzelnen Lesebefehl lassen sich dann alle Zeichen bis zum nächsten »CR/LF« einlesen. Deswegen dürfen die einzelnen Datensätze auch unterschiedliche Längen aufweisen.

In einer relativen Datei hingegen sind nur Daten von konstanter Länge erlaubt, dafür kann aber beliebig auf die einzelnen Datensätze zugegriffen werden. Soll beispielsweise in einer sequentiellen Datei der dreißigste Datensatz gelesen werden, so müssen erst die vorausgehenden 29 Datensätze gelesen werden, ehe der dreißigste verfügbar ist. Bei einer relativen Datei dagegen läßt sich der dreißigste Datensatz über die Satznummer direkt bestimmen, ohne andere Datensätze vorher lesen zu müssen.

Damit auf einen Datensatz bei einer relativen Datei direkt zugegriffen werden kann, muß der Computer in der Lage sein, den Datensatz auf der Diskette oder Festplatte zu lokalisieren.

Das geht aber nur, indem die Länge eines einzelnen Datensatzes bei einer relativen Datei festgelegt wird. Wenn dann der 30. Datensatz gelesen werden soll, kann der PC die Position dieses Datensatzes ermitteln, indem er das Dreißigfache der Satzlänge bildet. An dieser Stelle in der Datei beginnt dann der dreißigste Datensatz.

Beim PC werden die Daten einer Datei nicht direkt von der Diskette oder Festplatte gelesen oder auf sie geschrieben. Vielmehr richtet das Betriebssystem Pufferbereiche ein, die zum direkten Datenaustausch dienen. Dadurch wird der Dateizugriff erheblich beschleunigt.

Der erste Befehl, den man benötigt, um mit Dateien zu arbeiten, ist das Kommando zum Öffnen einer Datei. Bevor Sie eine Datei verwenden, müssen Sie ihr mit dem Open-Befehl einen Namen und eine Nummer zuweisen. Die Syntax des Befehls lautet:

```
open <Dateiname> [for <Modus>] as
# <Dateinummer>
[,len=<Satzlänge>]
```

Der Dateiname muß den Richtlinien von MS-DOS entsprechen. Für den Namen stehen daher maximal acht und für die Dateierweiterung drei Zeichen zur Verfügung. Der Modus gibt an, für welche Operationen die Datei verwendet werden soll. Diese Modi sind nur für die Verwendung im Zusammenhang mit sequentiellen Dateien von Bedeutung. Zur Wahl stehen die Modi »output«, »input« und »append«. Mit Ihnen lassen sich Datensätze in eine Datei schreiben (»output«), aus einer Datei lesen (»input«) oder an eine bestehende Datei anhängen (»append«).

Die Dateinummer dient zur Auswahl der Datei für spätere Lese- oder Schreibzugriffe. Hier kommt ein Wert zwischen 1 und 15 zur Anwendung. Der letzte Parameter schließlich ist optional, das heißt, er muß nicht mit angegeben werden (deshalb ist er auch eckig eingeklammert). Er legt die Satzlänge einer relativen Datei fest. Hier ist ein Wert zwischen 1 und 32767 erlaubt. Wird der Len-Parameter im Open-Befehl angegeben, entfällt die Angabe des Modus für den Schreibzugriff, weil bei relativen Dateien sowohl geschrieben als auch gelesen werden darf. Entweder öffnet man eine Datei als sequentielle Datei, dann entfällt »len« oder man öffnet

sie als relative Datei, dann entfällt »for«. Typische Open-Befehle sind also:

```
open "daten.dat" for output as #1
'Öffne sequentielle Datei
open "daten.dat" as #1, len=100
'Öffne relative Datei
```

Das Gegenstück zum Open-Befehl bildet der Close-Befehl. Er schließt eine geöffnete Datei. Die Syntax lautet:

```
close# [<Dateinummer>] [, <Dateinummer>] ..
```

Als Dateinummer wird die Nummer einer geöffneten Datei übergeben. Der Close-Befehl sorgt automatisch dafür, daß alle möglicherweise noch im Datenpuffer befindlichen Daten auf die Diskette oder Festplatte geschrieben werden. Nur wenn eine Datei mit »close« ordnungsgemäß geschlossen wurde, ist gewährleistet, daß alle Daten auch tatsächlich auf den Datenträger übertragen wurden. Führen Sie den Befehl »close« ohne eine Dateinummer aus, dann werden alle offenen Dateien der Reihe nach geschlossen.

Die Anweisung »close« leert die Datenpuffer und sorgt für das ordnungsgemäße Schließen der Dateien

Zum Lesen aus und Schreiben in Dateien dienen die Basic-Kommandos »print#«, »write#«, »input#«, »line input#« und »input\$« bei sequentiellen und »put#« und »get#« (im Zusammenhang mit »lset« und »rset«) bei relativen Dateien. Da die Methodik der Ein- und Ausgabe bei sequentiellen und relativen Dateien völlig unterschiedlich ist, werden beide Verfahren separat dargestellt und jeweils durch eigene Beispielprogramme verdeutlicht.

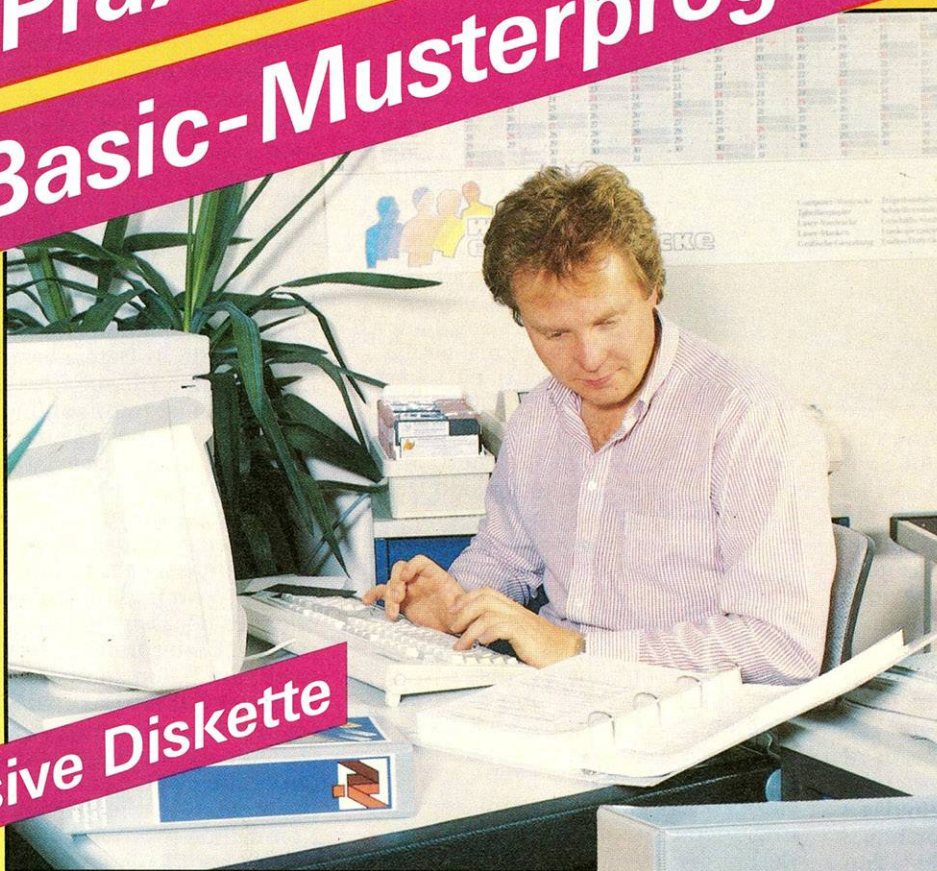
Die Ein- und Ausgabebefehle für die Arbeit mit sequentiellen Dateien sind sehr eng verwandt mit den von der »normalen« Ein- und Ausgabe bekannten Befehlen »print«, »write«, »input« und »input\$«. Die Syntax der Befehle lautet:

```
print# <Dateinummer>, <Ausdruck>
[,] [,] [<Ausdruck>] ..
write# <Dateinummer>, <Ausdruck>
[,] [,] [<Ausdruck>] ..
# <Dateinummer>, <Variable>
[, <Variable>] ..
<Variable>=input#(<Zeichenanzahl>,<Dateinummer>)
line input# <Dateinummer>,
<Variable>
```

Mit »print#« wird – wie vom Bildschirm gewohnt – der Inhalt einer Variablen oder ein unmittelbarer Wert in eine Datei geschrieben. Als Trennzeichen fungieren Semikolon oder Komma. Bei der Verwendung des Semikolons werden die Daten zweier Variablen direkt aneinander-

Praxiserprobte Basic-Musterprogramme

inklusive Diskette



Erst prüfen, dann kaufen:
Sehen Sie sich dieses Werk in
Ruhe an: 10 Tage lang dürfen
Sie Ihr Ansichtsexemplar
unverbindlich zu Hause prüfen.

Basic-Tools, Utilities und Anwenderprogramme für Wirtschaft, Technik, Mathematik

Dieses Nachschlagewerk bietet Ihnen:
auf ca. 1100 Seiten sofort einsetzbare Programme und Utilities für
PC-/MS-DOS-Computer. Hinweise auf gerätespezifische Besonderheiten ermöglichen Ihnen auch die problemlose Anpassung an
andere Rechner. Sie erhalten u. a.

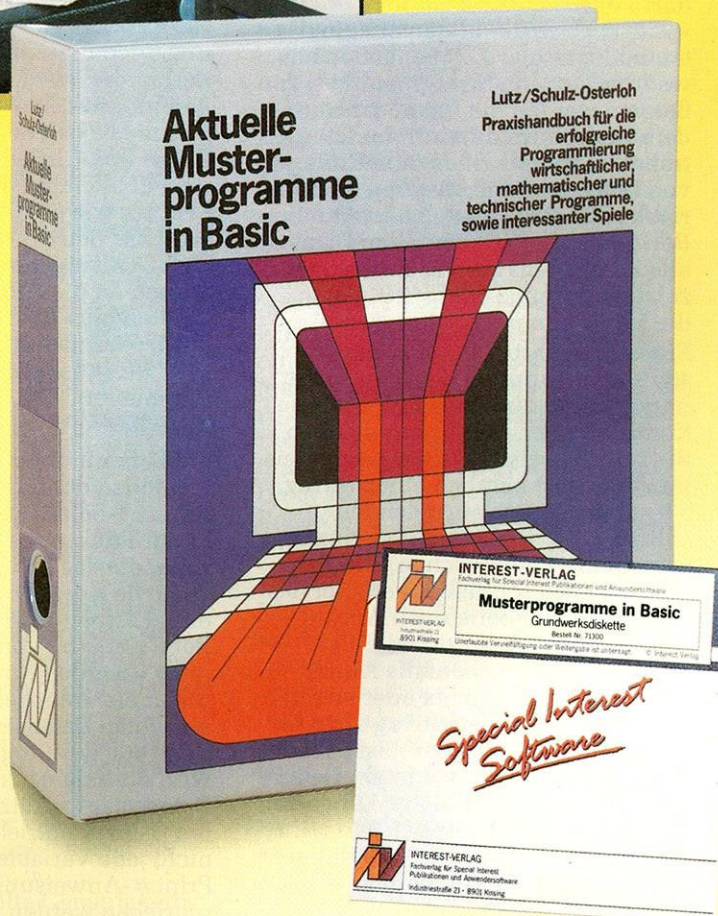
■ anspruchsvolle Anwenderprogramme wie

- **das leistungsstarke Textverarbeitungsprogramm**
„SOTEXT 4.0“, komplett menügesteuert und mit deutschen
Befehlen für den problemlosen Soforteinsatz
- **SOTEXT 4.0 zusätzlich als sofort lauffähige Version und in
Sourcecode auf Diskette!**
- **Programme für Wirtschaft und Verwaltung:** Finanzbuchhaltung „SOBUCH“, Netzplantechnik, Terminkalender, Berechnung der Einkommensteuer, Rabattermittlung, Tilgungsplan eines Darlehens, Restwertberechnung ...;
- **mathematische Programme:** Analyse von Funktionsgleichungen (z. B. Kurvendiskussion; lineare Algebra: z. B. Matrizenoperationen, Simplexalgorithmus);
- **statistische Programme** (u. a. Berechnung von Mittelwert, Streuung, Standardabweichung);
- **technische Programme:** Messen/Steuern/Regeln, Wertetabellen von Digitalschaltungen, Berechnen von Querschnittswerten ...;

Programme für die Ausbildung (z. B. Vokabeltraining);

■ Tools und vielseitige Routinen für

- **Grafik:** Grafik-Generator „SOGRAF“, Zeichengenerator;
 - **Dateiverwaltung und Dateioorganisation:** Dateigenerator, Komprimieren und Expandieren von Textdateien, Konvertieren von Dateien, Disketten-Editor „SOZAP“, indexsequentielle Dateioorganisation und Reorganisation, Hash-Dateioorganisation; Such- und Sortiervorgänge;
 - **Drucker** (z. B. Ausgabe von Random-Dateien mit Zeilenvorschub, Druckereinstellung);
 - **Bildschirm** (z. B. Bildschirm- und Menütechnik);
 - **Multitasking** (z. B. Programm zur Parallelverarbeitung);
- Jedes Programm ist ausführlich dokumentiert mit Problem-
beschreibung, Vorgehensweise, Struktogramm, Listing und
Probelauf.



Fordern Sie noch heute an:

Aktuelle Musterprogramme in Basic

stabiler Ringbuchordner, DIN A4, über 1100 Seiten,
inkl. Diskette „SOTEXT 4.0“, Bestell-Nr. 1300/71300,
Preis: DM 98,-.

Alle 2 – 3 Monate erhalten Sie Ergänzungsausgaben
zum Grundwerk mit je ca. 120 Seiten zum Seiten-
preis von 38 Pfennig (Abbestellung jederzeit möglich).



INTEREST-VERLAG
Industriestraße 21
D-8901 Kissing
Tel. 0 82 33/21 01-0


```
Ist schon eine Adreßdatei vorhanden? n
(N)euere Daten eingeben, (D)aten drucken oder (A)bbrechen? n
```

```
Eingabe des 1 ten Datensatzes
```

```
Name:      Ungeheuer
Vorname:    Martin
Strasse:    Hans-Stießberger-Str. 3
Ort:        Haar
```

```
weiter? j
```

```
Eingabe des 2 ten Datensatzes
```

```
Name:      Lieblich
Vorname:    Thomas
Strasse:    Hans-Stießberger-Str. 3
Ort:        Haar
```

```
weiter?
```

**Eine einfache
Adreßverwaltung
als Beispiel für das
Arbeiten mit sequen-
tiellen Dateien**

gereiht, so daß eine spätere Unterscheidung nicht mehr möglich ist. Bei der Verwendung des Kommas dagegen werden die Daten auch innerhalb der Datei durch ein Komma getrennt. Wie bei der Verwendung von »print« bei der Ausgabe auf dem Bildschirm, ist der Befehl auch um ein »using« zu erweitern. Dadurch lassen sich Daten in festgelegten Formaten in eine Datei ausgeben.

Der Befehl »write #« ist eine besondere Form des Kommandos »print #«. »write« allgemein gibt Daten inklusive der Zeichenketten aus, wenn es sich um Strings handelt. Außerdem werden die Daten auch auf dem Bildschirm durch Kommata getrennt. »input #« dient zum Einlesen von Daten aus einer Datei in eine Variable. Dabei werden ab der aktuellen Position in der sequentiellen Datei alle Zeichen bis zum nächsten Trennzeichen (Komma oder »CR/LF«) in die Variable eingelesen.

Einen erweiterten Input #-Befehl stellt »line input #« dar. Er gelangt immer dann zur Anwendung, wenn Kommata oder Anführungszeichen Bestandteile eines Strings sind. »line input #« liest nämlich einen String bis zum nächsten auftretenden »CR/LF« ein, akzeptiert also Kommata nicht als Trennzeichen. Die Funktion »input\$« schließlich erlaubt das Auslesen einer beliebigen Anzahl Zeichen eines Datensatzes. Mit »input\$« können ebenfalls Anführungsstriche, Kommata oder andere Trennzeichen in den String eingelesen werden. Die Funktion ist auch dann sehr hilfreich, wenn ein Datensatz die maximal zulässige Länge von 256 Zeichen überschreitet. Mit Hilfe des Befehls »input #« ist der Datensatz in diesem Fall nicht mehr vollständig lesbar.

Listing 1 zeigt eine typische Anwendung für eine sequentielle Datei. Das Programm erfährt bis zu 100 Adressen (Bild). Diese Daten werden in den Feldvariablen »nname«, »vname«,

»strasse« und »ort« gespeichert. Für die dauerhafte Speicherung auf Diskette oder Festplatte legt das Programm Seqfile eine sequentielle Datei namens »adress.dat« an. Zu Beginn fragt »Seqfile« daher, ob schon eine Adreßdatei auf Datenträger besteht. In diesem Fall wird sie zuerst eingelesen. Dann entscheidet der Anwender, ob er neue Daten eingeben, die Datensätze ausdrucken oder die Arbeit abbrechen will. Nach der Eingabe neuer Datensätze wird

»adress.dat« automatisch neu sortiert und wieder gespeichert. Besondere Beachtung verdienen die Befehle zur Dateiverwaltung im Programm. Neben den gewöhnlichen Open- und Close-Befehlen interessieren vor allem die Befehle, die den Drucker wie eine Datei ansteuern. Weil das Betriebssystem MS-DOS jedem Gerät eigene Gerätenamen zuordnet, ist die Ausgabe zu solchen Geräten über die Datei-Befehle von Basic vorzunehmen. Listing 1 (Zeile 510) weist dem Drucker mit dem Kommando

```
open "prn" for output as #1
```

die Dateinummer 1 zu und sendet im folgenden dann Daten mit dem Print #1-Befehl an den Drucker. Einen Trick enthält auch die Befehlszeile, die das Schreiben der Datensätze in die Adreßdatei der Diskette oder Festplatte erledigt (Zeile 460). Beim Einlesen von Daten mit »input #« werden Kommata vom Interpreter als Trennzeichen zwischen einzelnen Datensätzen verstanden. In Zeile 460 des Programms »Seqfile« wird deshalb nach jedem Feld eines Datensatzes zusätzlich ein Komma in die Datei geschrieben. Dadurch muß nicht jede Variable mit einer eigenen Print #-Anweisung in die Datei geschrieben werden.

Für viele Anwendungen sind sequentielle Dateien zu unflexibel und vor allem viel zu langsam im Zugriff. Probleme dieser Art lassen sich mit relativen Dateien beheben. Hier ist

der Zugriff nicht sequentiell, also hintereinander, sondern vielmehr direkt. Natürlich fordern relative Dateien auch einen hohen Grad an Organisation. Basic stellt dafür die folgenden Befehle zur Verfügung:

```
field# <Dateinummer>, <Länge1> as
<Feld1> [, <Länge2> as <Feld2> >]
get# <Dateinummer> [, <Satznummer>]
<Var>=loc(<Dateinummer>)
lset <Feld> = <Ausdruck>
put# <Dateinummer> [, <Satznummer>]
rset <Feld> = <Ausdruck>
```

Bevor mit den Befehlen eine relative Datei verwirklicht wird, muß das Programm genau festlegen, wie die Struktur dieser Datei auszusehen hat. Ein Direktzugriff auf alle Datensätze einer Datei ist nur dann möglich, wenn alle Datensätze eine konstante Länge besitzen. Diese Satzlänge wird mit dem Open-Befehl festgelegt. Der Interpreter legt dann im Speicher einen Bereich an, der genau einen solchen Datensatz aufnimmt.

Immer wenn in die Datei geschrieben oder aus der Datei gelesen wird, benutzt der Interpreter diesen Datenbereich, um Daten zu übergeben oder zu übernehmen. Das Problem besteht für das Programm also nur darin, die Daten in diesen Bereich zu schreiben beziehungsweise aus ihm zu lesen. Dazu dient der Field-Befehl. Dieses Kommando reserviert bestimmte Feldnamen für Teile des Datensatzbereichs des Interpreters.

Diese Feldnamen sind besondere Variablen, deren Inhalte sich im Gegensatz zu »normalen« Variablen immer an derselben Stelle im Speicher befinden. An einem praktischen Beispiel läßt sich der Umgang mit diesen Feldnamen gut verdeutlichen. Nehmen Sie einmal an, Sie möchten das Telefonbuch elektronisch erfassen und für die einzelnen Felder die folgenden Feldlängen einrichten:

```
Nachname 25
Vorname 20
Beruf 30
Strasse 30
Telefon 10
```

Insgesamt werden von einem Datensatz also 115 Byte belegt. Sie müssen beim Open-Befehl demnach mit einer Satzlänge von 115 arbeiten. Der Field-Befehl dient dazu, die einzelnen Felder eines Datensatzes zu nutzen. Damit erzeugen Sie symbolische Feldnamen, mit denen der Datensatzpuffer des Interpreters adressiert werden kann. Verwenden Sie für die Felder die gleichen Namen wie bei der Aufschlüsselung der Feldlängen:

```
field#1,25 as nachname$,20 as
vorname$,30 as beruf$,30 as strasse$,10
as telefon$
```

Damit sind die Variablen »nachname\$«, »vorname\$«, »beruf\$«, »strasse\$« und »telefon\$« als Feldvariablen einer relativen Datei definiert.

Beachten Sie bitte, daß die Variablen nur solange gültig sind, wie die relative Datei geöffnet ist.

Der Unterschied zu »normalen« Variablen besteht darin, daß der Inhalt der Feldvariablen immer an der gleichen Stelle im Speicher zu finden ist, nämlich im Puffer für den Datensatz, während gewöhnliche Variablen bei Neuzuweisungen in der Regel ihre Position im Speicher verändern. Klar ist deshalb auch, daß man mit den Feldvariablen nicht genauso arbeiten kann wie mit anderen Variablen. Dann würde nämlich bei einer Wertzuweisung an eine Feldvariable der Inhalt der neuen Variablen nicht mehr an der gleichen Stelle im Speicher stehen. Basic bietet deshalb zwei eigene Befehle, die eine Zuweisung von Daten an Feldvariablen erlauben. Sie heißen »lset« und »rset« und ordnen einer Feldvariablen einen Ausdruck entweder links- oder rechtsbündig zu. Dabei werden Leerzeichen automatisch ergänzt und die Feldvariable so auf die festgelegte Länge gesetzt.

Feldvariablen sind eine besondere Gattung von Variablen in Basic

Für die umgekehrte Richtung einer Zuweisung sind Feldvariablen wie gewöhnliche Variablen zu verwenden, weil hier das Problem der Verschiebung im Speicher niemals auftritt. Mit

```
a$=nachname$
```

läßt sich eine Feldvariable daher problemlos zu einem Funktionsargument machen. Bleibt noch das Problem, einen Datensatz auf die Diskette oder Festplatte zu schreiben beziehungsweise aus ihr zu lesen. Dazu dienen die Kommandos »put #« und »get #«, die jeweils genau einen Datensatz auf die Diskette speichern (»put #«) oder aus ihr lesen (»get #«). Nehmen Sie an, Sie hätten Ihre Telefonbuchdatei geöffnet und die Feldnamen wie oben festgelegt. Nun haben Sie mit der Sequenz

```
lset nachname$="Inkognito"
lset vorname$="Ingo"
lset beruf$="Inspektor"
lset strasse$="Ichweißnichtweg 11"
lset telefon$="123456"
```

die Daten eines Datensatzes in den Puffer übertragen. Durch Verwendung des Befehls »put # 1,1« werden diese Daten nun als Datensatz 1 in die relative Datei geschrieben. Mit »put # 1,100« wäre es Datensatz 100 geworden.

Den umgekehrten Weg beschreitet »get #«. Dieser Befehl überträgt einen beliebigen Datensatz in den Puffer. »get # 1,1« liest den ersten Datensatz, »get # 1,100« den hundertsten. Nach einem lesenden Zugriff auf eine rela-

tive Datei tragen die Feldvariablen die Inhalte der einzelnen Felder des Datensatzes und lassen sich nun direkt ausgeben oder anderen Variablen übergeben.

Für die Arbeit mit relativen Dateien verwendet Basic intern einen Dateizeiger, der immer auf den aktuellen Datensatz zeigt. Geben Sie bei »put #« oder »get #« keine Satznummer an, so schreibt beziehungsweise liest Basic den aktuellen Datensatz. Nach dem Öffnen einer Datei wird der Dateizeiger auf 1 gesetzt. Jedes Kommando »put #« oder »get #« erhöht nun diesen Zeiger. Wird also eine Satznummer nicht explizit angegeben, so läßt sich eine relative Datei auch sequentiell lesen.

Falls Sie den aktuellen Dateizeiger auslesen wollen, müssen Sie dazu den Loc-Befehl verwenden. Er gibt den Wert des Dateizeigers einer beliebigen offenen Datei an. Mit

```
zeiger=loc(1)
```

weisen Sie der Variablen »zeiger« den Wert des Dateizeigers der Datei 1 zu.

Mit diesen Kenntnissen sind Sie bereits in der Lage, eine relative Datei aufzubauen. Dabei wird aber sicher eine wichtige Frage auftauchen: Wie kann man Zahlenvariablen lesen und speichern?

Prinzipiell werden in relativen Dateien nur Zeichenketten (strings) verarbeitet. Natürlich läßt sich mit Hilfe der Str\$-Funktion jeder numerische Wert erst in eine Zeichenkette umwandeln und anschließend in der Datei speichern. Doch dabei wird der Platz auf der Diskette sehr unökonomisch verwendet.

Um dieses Problem zu meistern, stehen in Basic einige Funktionen zur Verfügung, die numerische Daten in komprimierte Zeichenketten umwandeln. Es sind dies die Funktionen

```
<Var>=mki$(<Ausdruck>)
<Var>=cvi(<Var>)
<Var>=mks$(<Ausdruck>)
<Var>=cvs(<Var>)
<Var>=mkd$(<Ausdruck>)
<Var>=cvd(<Var>)
```

Jede Mk-Funktion hat genau eine korrespondierende Cv-Funktion. Dabei stehen die Abkürzungen »i«, »s« und »d« hinter den beiden ersten Buchstaben für:

i (»Integer«, Ganzzahl; Kennzeichnung: »%«): belegt als Zeichenkette 2 Byte;

s (»Single«, einfache Genauigkeit; Kennzeichnung: keine): belegt als Zeichenkette 4 Byte;

d (»Double«, doppelte Genauigkeit; Kennzeichnung: »#«): belegt als Zeichenkette 8 Byte.

Die Funktionen sind somit für die drei unterschiedlichen numerischen Datentypen ausgelegt. Wollen Sie

beispielsweise eine Variable doppelter Genauigkeit in eine Zeichenkette umwandeln, so geschieht das wie folgt:

```
doppelt#=1.26578224567
doppelt$=mkd$(doppelt#)
```

Für die Rückkonvertierung nehmen Sie dann die Zeile

```
doppelt#=cvd(doppelt$)
```

Mit Hilfe der sechs unterschiedlichen Konvertierungsbefehle (für drei verschiedene Datentypen) lassen sich numerische Werte in ein stark komprimiertes Format umwandeln und in einer relativen Datei unterbringen. Tatsächlich verhält es sich allerdings nicht so, daß der Interpreter die Daten mit den oben genannten Funktionen umwandelt. Vielmehr ist das beschriebene Format genau dasjenige, welches Basic intern zur Speicherung der einzelnen Variablentypen verwendet.

Mit den Funktionen zur Konvertierung wird vielmehr erreicht, daß die Daten nicht – wie beispielsweise vor der Bildschirmausgabe – in das »gewohnte« Format umgerechnet werden. Statt dessen legt der Interpreter das interne Datenformat ungeändert in einer Zeichenkette ab. Natürlich sind die Konvertierungsbefehle auch unabhängig von relativen Dateien anwendbar. Sinnvoll sind die Funktionen ebenfalls zur Datenfernübertragung einzusetzen.

Zum Schluß unseres kleinen Ausflugs in die Welt der Dateien unter Basic soll auf eine Funktion hingewiesen werden, die sowohl für relative als auch sequentielle Dateien sinnvoll ist:

```
<Var>=lof(<Dateinummer>)
```

Diese Funktion ermittelt die Länge einer Datei in Byte. Bei relativen Dateien ermittelt sie mit einem Trick die Anzahl der gespeicherten Datensätze. Dazu wird lediglich der mit »lof« ermittelte Wert durch die Satzlänge der Datei geteilt.

Bei sequentiellen Dateien bestimmt eine Funktion direkt das Dateiende:

```
<Var>=eof(<Dateinummer>)
```

Die Eof-Funktion läßt sich in Schleifen-Konstrukten hervorragend dazu verwenden, bis zum Ende einer Datei alle Datensätze zu lesen. »eof« liefert dabei den Wahrheitswert wahr (-1), wenn das Dateiende erreicht ist, ansonsten aber den Wahrheitswert falsch (0). Meist wird »eof« in Schleifen folgender Form verwendet:

```
while not(eof(1))
..
.. '- hier stehen Befehle
.. '- zum Lesen der Datei
..
wend
```

Indem Sie nun selbst beginnen, mit Dateien zu arbeiten, vertiefen Sie Ihre Kenntnisse durch die Praxis. Oft

wird es dann zum Problem, alle Daten, die Sie in eine Datei hineingeschrieben haben, auch wieder herauszuholen.

Das Programm »getall.bas« (Listing 2) verschafft Ihnen einen Einblick, der bis in die letzten Winkel Ihrer Dateien reicht. Einerseits demonstriert »getall.bas« den Aufbau einer relativen Datei, und andererseits lassen sich mit dem Programm sämtliche

Dateien Byte für Byte ansehen. Dabei darf es sich um Texte, Programme, sequentielle oder relative Dateien handeln. Probieren Sie es einmal an der von »Seqfile« erzeugten Adreßdatei »adress.dat« aus.

Zum Lesen benutzt »Getall« einen Trick: Bei sequentiellen Dateien werden Werte (Daten) nur bis zum nächsten Trennzeichen gelesen (außer bei der Verwendung der Input\$-Funk-

tion). Zur Umgehung dieser Einschränkung öffnet »Getall« die zu lesende Datei als relative Datei mit der Satzlänge 1 und übernimmt dann mit jeder Put #-Anweisung genau ein Zeichen aus der Datei.

Um die Bildschirmausgabe zu unterbrechen, genügt ein Antippen der Leertaste.

(Martin Althaus/zi)

```
[CC5] 1 REM Programmname: SEQFILE.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM      DOS International
[LA7] 10 DIM NNAME$(100),VNAME$(100),STRASSE$(100),ORT$(100)
[R83] 15 'Variablen anlegen
[PB2] 20 CLS:ANZAHL=0
[XP8] 25 'Bildschirm löschen und Satzanzahl zurücksetzen
[YC9] 30 INPUT "Ist schon eine Adreßdatei vorhanden? ",SCHON$
[FB8] 40 IF LEFT$(SCHON$,1)<>"J" AND LEFT$(SCHON$,1)<>"j" THEN 130
[B41] 45 'nein
[OH5] 50 OPEN "adress.dat" FOR INPUT AS #1
[HK2] 60 INPUT#1,ANZAHL
[HX3] 65 'Satzanzahl einlesen
[RT2] 80 FOR I=1 TO ANZAHL
[PK7] 90 INPUT#1,NNAME$(I),VNAME$(I),STRASSE$(I),ORT$(I)
[Y02] 95 'einlesen
[CG1] 100 NEXT I
[9X3] 105 'und Schleife beenden
[WL1] 110 CLOSE 1
[6M4] 115 'Datei wieder schließen
[F33] 120 ANZAHL=ANZAHL+1
[V74] 125 'Satzanzahl justieren
[T46] 130 WHILE AUSWAHL$<>"A" AND AUSWAHL$<>"a"
[V75] 135 'wenn nicht abgebrochen wird
[CLC] 140 INPUT "(N)eue Daten eingeben, (D)aten drucken oder (A)bbrechen? ",AUSWAHL$
[015] 145 'eine Funktion auswählen
[OL7] 150 IF AUSWAHL$="N" OR AUSWAHL$="n" THEN GOSUB 190
[KH4] 155 'Neueingabe ausführen
[VH7] 160 IF AUSWAHL$="D" OR AUSWAHL$="d" THEN GOSUB 510
[4R3] 165 'Drucken ausführen
[071] 170 WEND
[SH4] 175 'Schleife abschließen
[E31] 180 END
[MA3] 185 'Programm beenden
[6X5] 190 REM ----- Neue Daten eingeben -----
[SC4] 200 IF ANZAHL=100 THEN RETURN
[4A6] 205 'kein Platz für neue Datensätze
[UA2] 210 ANTWORT$=""
[UK3] 215 'Antwort löschen
[2B6] 220 WHILE ANTWORT$<>"N" AND ANTWORT$<>"n"
[R66] 225 'falls weitere Eingabe gewünscht
[S43] 230 ANZAHL=ANZAHL+1
[R24] 235 'Satzanzahl erhöhen
[YP1] 240 PRINT
[VX7] 250 PRINT "Eingabe des ";ANZAHL;"ten Datensatzes"
[4R1] 260 PRINT
[CN5] 270 INPUT "Name: ",NNAME$(ANZAHL)
[4W4] 275 'die einzelnen Felder des
[826] 280 INPUT "Vorname: ",VNAME$(ANZAHL)
[AF4] 285 'Datensatzes ausfüllen
[XS6] 290 INPUT "Strasse: ",STRASSE$(ANZAHL)
[CH5] 300 INPUT "Ort: ",ORT$(ANZAHL)
[RP1] 310 PRINT
[BX4] 320 INPUT "weiter? ",ANTWORT$
[EA5] 325 'weitere Eingabe gewünscht?
[S61] 330 WEND
[NX4] 340 REM -- nun noch sortieren ---
[2H3] 350 FOR I=1 TO ANZAHL-1
[8G6] 355 'sortiere die Daten mit Bubble-Sort
[7H3] 360 FOR J=1 TO ANZAHL-1
[L85] 370 IF NNAME$(J+1)>NNAME$(J) THEN 420
[A06] 375 'wenn Reihenfolge nicht korrekt
[YC4] 380 SWAP NNAME$(J+1),NNAME$(J)
[9S5] 385 'dann tausche die Variablen aus
[HE4] 390 SWAP VNAME$(J+1),VNAME$(J)
[275] 400 SWAP STRASSE$(J+1),STRASSE$(J)
[TR3] 410 SWAP ORT$(J+1),ORT$(J)
```

```
[MS1] 420 NEXT J,I
[GL6] 425 'Sortierschleifen abschließen
[HP4] 430 '----- und abspeichern -----
[WV5] 440 OPEN "adress.dat" FOR OUTPUT AS #1
[WV5] 445 'seq. Datei für Ausgabe öffnen
[TV2] 450 PRINT#1,ANZAHL
[BK4] 455 'erst Satzanzahl ablegen
[W63] 460 FOR I=1 TO ANZAHL
[2C7] 465 'in einer Schleife alle Sätze speichern
[CO9] 470 PRINT#1,NNAME$(I),"",VNAME$(I),"",STRASSE$(I),"",ORT$(I)
[AH1] 480 NEXT I
[4B3] 485 'dann Schleife und
[UM1] 490 CLOSE 1
[4N4] 495 'Datei wieder schließen
[0M1] 500 RETURN
[N65] 505 'zurück zum Hauptprogramm
[LH4] 510 '----- Daten drucken -----
[UH4] 520 OPEN "prn" FOR OUTPUT AS #1
[CR1] 530 PRINT#1,
[028] 540 PRINT#1,"Adreßliste von ";ANZAHL;" Adressen"
[M43] 550 PRINT#1,:PRINT#1,
[Y63] 560 FOR I=1 TO ANZAHL
[M25] 570 PRINT#1,VNAME$(I);" ";NNAME$(I)
[CP3] 580 PRINT#1,STRASSE$(I)
[NY2] 590 PRINT#1,ORT$(I)
[5R1] 600 PRINT#1,
[TG1] 610 NEXT I
[9M1] 620 CLOSE 1
[BM1] 630 RETURN
```

Listing 1. Mit »seqfile.bas« erfassen Sie bis zu 100 Adressen

```
[115] 1 REM Programmname: GETALL.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM      DOS International
[5A4] 10 INPUT "Dateiname? ",DNAME$
[MT2] 15 'Namen einlesen
[8U0] 20 CLS
[ZM3] 25 'Bildschirm löschen
[WK3] 30 OPEN DNAME$ AS #1 LEN=1
[E66] 35 'Datei öffnen mit der Satzlänge 1
[813] 40 FIELD#1,1 AS BYTE$
[126] 45 'in BYTE$ werden die Daten abgelegt
[7A2] 50 LAENGE=LOF(1)
[955] 55 'die Länge der Datei ablegen
[4U2] 60 FOR I=1 TO LAENGE
[AX5] 65 'in einer Schleife alle Bytes lesen
[8C3] 70 IF INKEY$="" THEN 130
[916] 75 'falls Leertaste gedrückt: Pause
[A01] 80 GET#1
[NA3] 85 'Zeichen auslesen
[XB2] 90 PRINT BYTE$;
[255] 95 'auf dem Bildschirm ausgeben
[CG1] 100 NEXT I
[Y65] 105 'und Schleife abschließen
[WL1] 110 CLOSE 1
[125] 115 'dann die Datei schließen
[W21] 120 END
[0L4] 125 'und das Programm beenden
[S13] 130 WHILE INKEY$<>" "
[S86] 135 'Schleife: Solange nicht wieder die
[R61] 140 WEND
[1R6] 145 'Leertaste gedrückt wird, tue nichts!
[3N1] 150 GOTO 80
[796] 155 'sonst zurück in die Leseschleife
```

Listing 2. »getall.bas« gewährt Ihnen Einblick in Dateien - Byte für Byte

Rekursion und GW-Basic

Als Basic-Programmierer stehen Sie im allgemeinen vor dem Problem, viel von Rekursion zu hören, aber wenig damit anfangen zu können. Dieser Beitrag hilft Ihnen dabei, die Vorgänge bei der Rekursion zu verstehen und diese in Basic nachzubilden beziehungsweise in Basic zu übertragen.

Zunächst einmal soll der Begriff »Rekursion« erläutert werden: In der Mathematik bedeutet er das »Zurückgehen bis zu bekannten Werten« oder die »Definition mit Hilfe der Struktur, die definiert werden soll«. Etwas allgemeiner wird von der »Bildung von unendlich vielen Sätzen nach bestimmten Regeln, wobei die einzelnen Einheiten jeweils wieder Sätze sind« gesprochen. Das hört sich alles zunächst recht kompliziert an. Kürzer sagt es ein alter Mathematikerwitz: »Rekursiv: siehe rekursiv«. Sie alle wurden sicher als Kind schon mit der Rekursion (in Form einer Geschichte) vertraut gemacht: »Es war einmal ein Vater, der hatte sieben Söhne. Einer der Söhne sagte zu seinem Vater: ›Erzähle uns doch einmal eine Geschichte‹, und so fing der Vater an:

Es war einmal ein Vater, der hatte sieben Söhne. Einer der Söhne sagte zu seinem Vater...«

An diesem Beispiel läßt sich bereits ein Problem der Rekursion ablesen. Versuchen Sie es mit dem folgenden Programm:

```
100 gosub 200
110 end
200 i=i+1
210 print i,"ter Aufruf"
220 gosub 200
230 return
```

Dieses Programm entspricht der Geschichte, weil es sich immer wieder selbst aufruft (Zeile 220). Nach sehr kurzer Zeit erhalten Sie jedoch die Fehlermeldung »Out of Memory«. Das Programm hat nämlich einen entscheidenden Nachteil: Es bricht nie ab.

Mit rekursiven Formulierungen lassen sich auch große Strukturen einfach und elegant beschreiben

Das Abbrechen (Terminieren) eines Programms ist aber eine Grundvoraussetzung für das Arbeiten mit solchen Strukturen. Weiter erkennen Sie aber auch die Vorteile einer rekursiven Formulierung: Mit wenigen Zeilen werden »unendliche« Strukturen beschrieben; eine Eigenschaft, die mit Schleifen-Konstrukten (also einer iterativen Lösung) doch sehr schwerfällig und unleserlich werden kann.

Vollziehen Sie eine kleine Änderung an Ihrem Programm, und schon funktioniert es:

```
220 if i < 10 then gosub 200
```

Dies ist Ihre erste Rekursion in Basic. Das Unterprogramm ab Zeile 200 ruft sich selbst neunmal auf, bis die Endbedingung (»i < 10«) erreicht ist und die aufgerufenen Unterprogramme beendet werden. Warum aber gab der Computer bei dem ersten Programm die Fehlermeldung »Out of Memory« aus, und wie schafft er es, immer wieder an die richtige Stelle des Programms zurückzuspringen, ohne durch die wiederholten Aufrufe durcheinanderzukommen?

Der Computer merkt sich auf dem sogenannten Unterprogrammstack (Stapelspeicherbereich), von wo aus das Unterprogramm aufgerufen wurde. Da der Speicher für diesen Stack in Basic sehr beschränkt ist, erfolgt die Fehlermeldung ziemlich schnell. Die Funktionsweise des Stacks ist recht einfach; es handelt sich um einen sogenannten LIFO-Speicher (»Last in, first out« – Wer zuletzt kommt, geht als erster). Simulieren Sie die Funktionsweise des Programms – nur gestützt auf Ihren biologischen Computer Gehirn – auf Ihrem Schreibtisch. Nehmen Sie sich zehn Zettel, und führen Sie das Programm »von Hand« aus.

Indem Sie ein Programm schrittweise in Gedanken ausführen, wird das LIFO-Prinzip verständlicher

Zeile 100 enthält das erste »gosub«, das der Computer bearbeitet. Notieren Sie sich die Zeile 110 auf dem ersten Zettel (dies ist der erste Befehl, der nach dem »gosub« ausgeführt werden muß). Diesen Zettel legen Sie zur Seite (das erste Element des Stacks). Sie durchlaufen die Zeilen bis 220, wo das zweite »gosub« auftaucht. Also 230 auf den freien Zettel notieren und auf den ersten Zettel mit der Nummer 110 legen. Sie bilden auf diese Weise einen Stapel (Stack), womit der Begriff erklärt wäre. Ebenso verfahren Sie mit den anderen Zetteln, bis die Bedingung in Zeile 220 nicht mehr erfüllt ist. Nun gelangen Sie zum ersten »re-

turn«. Genauso, wie auch der Computer verfahren würde, nehmen Sie den oberen Zettel vom Stack und schauen nach, welche Zeile als nächstes ausgeführt werden muß (LIFO – der oberste Zettel wird als erster zurückgenommen). Danach werfen Sie den Zettel weg. Sie gelangen wieder zu einem »return« und behandeln die weiteren Zettel genauso, bis Sie zu Zeile 110 gelangen, die dann das Programm beendet. Sie müssen sich nun klar machen, daß dieser Mechanismus in keiner Weise davon abhängt, welches »gosub« und welches »return« für die jeweilige Aktion verantwortlich ist. Warum also nicht auch wieder das gleiche Unterprogramm aufrufen?

Die Fachbegriffe für den Umgang mit dem Stack sollen hier der Vollständigkeit halber auch genannt werden:

Push – auf den Stack speichern,
Pop – vom Stack lesen.

Durch den rekursiven Aufruf einer Routine ersparen Sie sich eine Menge Programmierarbeit

Wie Sie bereits erfahren haben, ist der Stack begrenzt und daher auch nur eine bestimmte Verschachtelungstiefe von Unterprogrammen zu realisieren (die Verschachtelungstiefe gibt die maximale Anzahl aufeinanderfolgender Gosub-Befehle an). Ist Basic also doch für die Rekursion verwendbar? Leider trifft dies nur in dem bis hierher aufgezeigten Maße zu: Der Aufruf darf nicht zu oft erfolgen (Verschachtelungstiefe), und die verwendeten Variablen dürfen sich nicht gegenseitig überschneiden (wenn eine Variable verwendet wird, darf diese nicht ungewollt die vorherigen Werte dieser Variablen überschreiben).

In Sprachen wie zum Beispiel C oder Pascal ist dies alles unproblematisch, weil sie zum einen lokale Variablen erlauben (Variablen, die nur innerhalb des Unterprogramms gültig sind und daher bei jedem Aufruf des Unterprogramms auf dem Stack neu angelegt werden), die Übergabe von Parametern gestatten (vom Stack-Mechanismus her das gleiche wie die lokalen Variablen) und zum anderen einen wesentlich größeren Stack-Bereich zur Verfügung stellen (bei Standard-einstellung in Turbo Pascal 16 KByte).

Ein konkretes Beispiel (das Standardbeispiel zur Rekursion) mit Nachbildung eines Stacks zeigt das Programm »fak.bas« (Listing 1). Es handelt sich um die Berechnung der Fakultät. Mit der Fakultät von n (Schreibweise: n!) bezeichnet man

das Produkt der ersten n natürlichen Zahlen

(Beispiel: $6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$).

Zu programmieren ist folgende Vorschrift:

$n! = 1$, wenn $n < 2$
 $n \cdot (n-1)!$, wenn $n \geq 2$

An dieser Vorschrift erkennen Sie die Schwierigkeiten, die Basic mit der Rekursion hat.

Zur Verwaltung lokaler Variablen muß ein eigener Stack angelegt und verwaltet werden

Für die Fakultätsberechnung muß der aktuelle Wert von n gespeichert werden, bevor die Fakultät von $(n-1)$ berechnet wird. Er würde sonst überschrieben, und der Interpreter hätte keinen Zugriff mehr auf diesen Wert. Weil der Wertebereich realer Zahlen in Basic eingeschränkt ist, stellt der begrenzte Stack für Unterprogram-

maufrufe hier noch kein Problem dar. Um das korrekte Zurückkehren aus dem jeweiligen Unterprogramm kümmert sich Basic also allein. Für die Variable n muß ein eigener Stack angelegt und vom Programm verwaltet werden (Zeile 180 bis 290). Dieses Beispiel ist stark mathematisch »angehaucht« und auch nur zum Aufzeigen der Zusammenhänge gedacht. Nicht rekursiv programmiert ist es sogar ein Einzeiler:

```
100 zahl=1:for i=1 to n :
    zahl=zahl*i:next i
```

Ein zweites Beispiel zeigt auf, wie sich rekursive Programme in die Programmiersprache Basic übertragen lassen. Auch hierbei handelt es sich um ein Paradebeispiel für die Rekursion, nämlich Quicksort. Quicksort ist ein sehr schnelles Sortierverfahren, dessen Sortiermethode verblüffend einfach ist. Die folgende Beschreibung beschränkt sich auf eine Zahlenliste, die selbstverständlich

durch eine Liste von Zeichenketten (Strings) zu ersetzen ist.

- Wähle die Zahl in der Mitte der Liste als Referenzelement.
- Von links kommend, suche die erste Zahl, die größer als das Referenzelement ist.
- Von rechts kommend, suche die erste Zahl, die kleiner als das Referenzelement ist.
- Tausche die so gefundenen Zahlen aus.
- Wiederhole die Schritte 1 bis 4, bis die rechte und linke Grenze sich treffen.

Ein geradezu klassisches Beispiel für eine Rekursion – der Sortieralgorithmus »Quicksort«

Auf diese Weise läßt sich eine Liste erzeugen, die aus einer linken Liste besteht, die kleiner als das Referenz-

Sort (Links,Rechts)

i = Links	
j = Rechts	
Referenz = Zahl((Links+Rechts)/2)	
solange i <= j	
solange Zahl(i) < Referenz	
i = i + 1	
solange Referenz < Zahl(j)	
j = j - 1	
i < j	
nein	ja
tausche Zahl(i),Zahl(j)	
i = i + 1	
j = j - 1	
Links < j	
nein	ja
Sort(Links,j)	
i < Rechts	
nein	ja
sort(i,Rechts)	

↑
Bild 1. Struktogramm zu »Quicksort« – die rekursive Lösung

Bild 2. Struktogramm zu »Quicksort« – die iterative Lösung

Listen = 1	
Anfang(Listen) = 1	
Ende(Listen) = AnzahlElemente	
solange Listen <> 0	
Links = Anfang(Listen)	
Rechts = Ende(Listen)	
Listen = Listen - 1	
solange Links < Rechts	
Links1 = Links	
Rechts1 = Rechts	
Referenz = Zahlen((Links+Rechts)/2)	
solange links1 <= rechts1	
solange Zahlen(Links1) < Referenz	
Links1 = Links1 + 1	
solange Referenz < Zahlen(Rechts1)	
Rechts1 = Rechts1 - 1	
Links1 > Rechts1	
ja	nein
tausche Zahlen(links1), Zahlen(rechts1)	
Links1 = Links1 + 1	
Rechts1 = Rechts1 - 1	
Links1 >= Rechts	
ja	nein
Listen = Listen + 1	
Anfang(Listen) = Links1	
Ende(Listen) = Rechts	
Rechts = Rechts1	

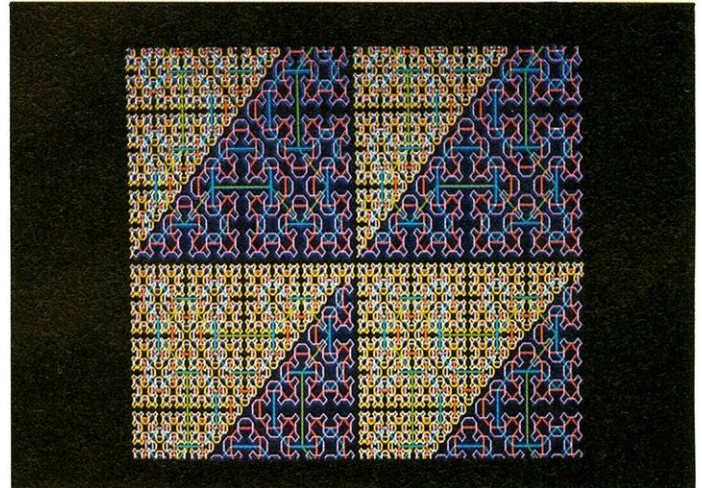
element ist, und einer rechten Liste, die nur Zahlen enthält, die größer als das Referenzelement sind. Nun ergeben sich praktisch zwei unsortierte Listen, auf die man die Schritte 1 bis 5 anwendet. So verfährt man, bis die Teilliste nur noch ein Element enthält. Sie sollten dieses Verfahren einmal auf dem Papier ausprobieren. Das Struktogramm (Bild 1) gibt die nötigen Schritte vor.

Auch in diesem Beispiel (Listing 2) legen Sie für jede aufgeführte lokale Variable und jeden aufgeführten Parameter einen Stack an (Zeile 150) und verwalten diesen Stack selbst (Zeile 410 und 450). Das Programm »quick.bas« erledigt das Sortieren von 1000 Zahlen in ungefähr sieben Minuten (Standard-PC mit 4,77 MHz Taktfrequenz). Dies ist bereits eine passable Zeit.

Nun geht es aber ans Optimieren. Man sollte Basic zwar nicht unbedingt etwas aufzwingen, wozu es nicht geschaffen ist; nachdem die Lösung aber erst einmal lauffähig ist, darf experimentiert werden. Betrachten Sie zunächst die Beschreibung des Algorithmus genauer. Die zwischengespeicherten Werte sind gar nicht alle unbedingt notwendig. Schließlich ist eine Liste ja jeweils weiter zu bearbeiten (beispielsweise immer erst die linke Liste). Das Programm muß also stets nur die übrigbleibenden Grenzen der Teilliste, die noch nicht sortiert ist, zusätzlich speichern. Somit ist ein wiederholter Aufruf des Programms nicht mehr notwendig.

Sie gelangen also auch hier, wie bereits bei der Fakultät, mit einer normalen Schleife (einer Iteration) ans Ziel. Das Ergebnis sehen Sie in Listing 3, die neue Struktur des Programms in Bild 2. Dieser Algorithmus ist in Basic wesentlich schneller: 1000 Zahlen werden jetzt in dreieinhalb Minuten (Standard-PC mit 4,77 MHz Taktfrequenz) sortiert. Der enorme Zeitgewinn basiert auf der Ein-

Bild 3. Sierpinski-Kurven veranschaulichen die Rekursion grafisch



sparung der Unterprogrammverwaltung durch Basic und der Stackverwaltung durch Sie selbst (respektive Ihr Programm).

Häufig läßt sich ein rekursives Problem so einfach wie hier gezeigt auf eine iterative Lösung zurückführen. Nicht nur Einsteigern bereitet das Übertragen rekursiver Definitionen in Basic erhebliche Schwierigkeiten. Der hier aufgezeigte Weg soll Ihnen helfen, die einzelnen Schritte zu vereinfachen und nach einem bestimmten Schema vorzugehen. Für das Umsetzen der Probleme muß aber ein gewisses Verständnis für Rekursionen vorhanden sein, das Sie nun sicher erhalten haben.

Der iterative Weg führt in GW-Basic wesentlich schneller ans Ziel als eine Rekursion

Zum Abschluß ein Beispiel für ein Programm, das durch den Einsatz einer Iteration anstelle einer Rekursion kaum mehr verständlich wäre: Das Zeichnen von Sierpinski-Kurven rekursiv in Basic (Listing 4). Die Sierpinski-Kurve besteht aus einem Grundmuster, wobei beim wiederholten Aufruf jede Linie des Grundmu-

sters durch das Muster selber ersetzt ist (Bild 3). Die Darstellung einer derartigen Kurve wird nur durch die Auflösung des Bildschirms und die Stack-Tiefe von Basic begrenzt.

Zum Schluß eine Aufgabe: Organisieren Sie Ihre Daten doch in Form einer Baumstruktur. Ein (Wurzel-) Element besitzt zwei untergeordnete Elemente; ein linkes (kleineres) und ein rechtes (größeres) Element. Jedes dieser beiden Elemente hat wiederum zwei untergeordnete Elemente und so weiter. Es entsteht eine an die Äste eines Baumes erinnernde Anordnung, die der Struktur ihren Namen gab. Zum Aufbau des Baumes benötigen Sie ein rekursives Unterprogramm:

```
Element_Einfügen(Zeiger)
wenn Zeiger leer, dann Zeiger = Element
sonst wenn Element < Zeiger, dann
Element_Einfügen (linker Zeiger) sonst
Element_Einfügen(rechter Zeiger)
```

Für die Realisierung der Aufgabe sollten Sie folgende Felder dimensionieren:

```
dim lzeiger%(1000), rzeiger%(1000),
daten$(1000)
```

Die Aufgabe ist gestellt, frisch ans Werk!

(Helmut Amshove/zi)

```
[YL4] 1 REM Programmname: FAK.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM      DOS International
[R65] 100 REM *****
[UX6] 110 REM * Simulation der Rekursion in *
[FG7] 120 REM * GWBASIC durch Unterprogrammaufruf *
[746] 130 REM * und Stack-Nachbildung *
[375] 140 REM *****
[Z97] 150 DIM STACK(24) : REM maximale Stackgröße
[NL1] 160 KEY OFF
[4A5] 170 GOTO 520 : REM Programmanfang
[TC5] 180 REM ***** P U S H *****
[H27] 190 IF STACK<24 THEN PRINT "Stacküberlauf":END
[EE7] 200 STACK=STACK+1 : ' Stackzeiger erhöhen
[FL7] 210 STACK(STACK)=STACK(0) : ' Zahl übernehmen
[ZT1] 220 GOSUB 390
[3M1] 230 RETURN
[SA5] 240 REM ***** P O P *****
[PN7] 250 IF STACK<1 THEN PRINT "Stackunterlauf":END
[904] 260 STACK(0)=STACK(STACK)
[PU7] 270 STACK=STACK-1 : ' Stackzeiger erniedrigen
[HU1] 280 GOSUB 390
[NM1] 290 RETURN
```

```
[AH5] 300 REM ***** F a k u l t ä t *****
[XA5] 310 IF ZAHL=1 THEN WERT=1 : GOTO 380
[8H2] 320 STACK(0)=ZAHL
[U54] 330 GOSUB 180 : ' Push Zahl
[WB2] 340 ZAHL=ZAHL-1
[FY6] 350 GOSUB 300 : ' berechne Fakultät(zahl-1)
[SZ3] 360 GOSUB 240 : ' Pop Zahl
[9H3] 370 WERT=WERT*STACK(0)
[MM1] 380 RETURN
[K26] 390 REM ***** Stack anzeigen *****
[S53] 400 FOR I=1 TO STACK
[152] 410 LOCATE I,40
[UT2] 420 PRINT STACK(I)
[VG1] 430 NEXT I
[GH3] 440 FOR I=STACK+1 TO 24
[D52] 450 LOCATE I,40
[5W3] 460 PRINT " ";
[7H1] 470 NEXT I
[GX3] 480 FOR I=1 TO 100 : NEXT I
[BB7] 490 LOCATE 15,1 : PRINT "Aktueller Wert: ",WERT
[0M1] 500 RETURN
[Z15] 510 REM ***** Programmanfang *****
```

Listing 1. »fak.bas« berechnet die Fakultät rekursiv in Basic


```
[P41] 520 CLS
[9S5] 530 PRINT "Berechnung der Fakultät"
[PC1] 540 PRINT
[ZV5] 550 INPUT "Fakultät von (<=24) ",ZAHN
[4C6] 560 GOSUB 300 : ' Aufruf Fakultät
[3M5] 570 LOCATE 15,1 : PRINT STRING$(42," ")
[RB5] 580 LOCATE 5,1 : PRINT "ist ";WERT
[T31] 590 END
```

Listing 1. »fak.bas« berechnet die Fakultät rekursiv in Basic

```
[TU4] 1 REM Programmname: QUICK.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[BZ3] 100 REM *****
[D74] 110 REM * Quick sort *
[H94] 120 REM * rekursiv *
[MZ3] 130 REM *****
[KS7] 140 REM Eigenen Stack für Variablen einrichten
[RN7] 150 DIM STACKL(20),STACKR(20),STACKI(20),STACKJ(20)
[Z65] 160 DIM A(1000) : ' Zahlenarray
[3F5] 170 GOTO 550 : ' Hauptprogramm
[5H4] 180 REM *****
[AH5] 190 REM Sortier routine
[FG4] 200 REM *****
[922] 210 ANZ=ANZ+1
[OK3] 220 PRINT "Aufruf ",ANZ
[575] 230 STACKI(STACKT)=STACKL(STACKT)
[R75] 240 STACKJ(STACKT)=STACKR(STACKT)
[DK7] 250 REFERENZ=A((STACKL(STACKT)+STACKR(STACKT))/2)
[AT6] 260 WHILE STACKI(STACKT)<=STACKJ(STACKT)
[G86] 270 WHILE A(STACKI(STACKT))<REFERENZ
[LVS] 280 STACKI(STACKT)=STACKI(STACKT)+1
[SC1] 290 WEND
[DY5] 300 WHILE REFERENZ<A(STACKJ(STACKT))
[BV5] 310 STACKJ(STACKT)=STACKJ(STACKT)-1
[SC1] 320 WEND
[MA7] 330 IF STACKI(STACKT)>STACKJ(STACKT) THEN 390
[CG4] 340 HILFE=A(STACKI(STACKT))
[B96] 350 A(STACKI(STACKT))=A(STACKJ(STACKT))
[N94] 360 A(STACKJ(STACKT))=HILFE
[806] 370 STACKI(STACKT)=STACKI(STACKT)+1
[K06] 380 STACKJ(STACKT)=STACKJ(STACKT)-1
[A71] 390 WEND
[627] 400 IF STACKL(STACKT)>=STACKJ(STACKT) THEN 460
[C23] 410 STACKT=STACKT+1
[EH5] 420 STACKL(STACKT)=STACKL(STACKT-1)
[ZH5] 430 STACKR(STACKT)=STACKR(STACKT-1)
[N36] 440 GOSUB 180 : ' Sortier routine
[Y23] 450 STACKT=STACKT-1
[V27] 460 IF STACKI(STACKT)>=STACKR(STACKT) THEN 520
[Y23] 470 STACKT=STACKT+1
[UH5] 480 STACKL(STACKT)=STACKI(STACKT-1)
[1L5] 490 STACKR(STACKT)=STACKR(STACKT-1)
[B36] 500 GOSUB 180 : ' Sortier routine
[M23] 510 STACKT=STACKT-1
[M22] 520 ANZ=ANZ-1
[YY5] 530 PRINT "Zurück zum ",ANZ," Aufruf"
[CM1] 540 RETURN
[E75] 550 REM *****
[AP4] 560 REM Hauptprogramm
[M75] 570 REM *****
[A55] 580 REM Zufallszahlen erzeugen
[RU6] 590 FOR I=1 TO 1000 : A(I)=INT(RND*1000) : NEXT I
[AP3] 600 PRINT "Sortieren ..."
[ZV1] 610 STACKT=1
[Z23] 620 STACKL(STACKT)=1
[ZG3] 630 STACKR(STACKT)=1000
[FP1] 640 AS=TIME$
[ZU1] 650 GOSUB 180
[XP1] 660 ES=TIME$
[Ø72] 670 PRINT AS,ES
[S31] 680 END
```

Listing 2. Nachbildung des rekursiven Quicksort: »quick.bas«

```
[G15] 1 REM Programmname: QUICKI.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[FC4] 100 REM *****
[HD3] 110 REM Q U I C K S O R T
[VR2] 120 REM iterativ
[SC4] 130 REM *****
```

```
[KW6] 140 REM Speicher für Grenzen der Listen
[CK3] 150 DIM ANFANG(20),ENDE(20)
[K75] 160 DIM ZAHLEN(1000) : ' Zahlenfeld
[VC5] 170 GOTO 530 : ' Hauptprogramm
[8T4] 180 REM *** Q U I C K S O R T *****
[TW1] 190 LISTEN=1
[852] 200 ANFANG(1)=1
[622] 210 ENDE(1)=1000
[913] 220 WHILE LISTEN<>Ø
[CM4] 230 PRINT "Stacktiefe ",LISTEN
[S24] 240 LINKS=ANFANG(LISTEN)
[7X3] 250 RECHTS=ENDE(LISTEN)
[P83] 260 LISTEN=LISTEN-1
[NX3] 270 WHILE LINKS<RECHTS
[PW2] 280 LINKS=LINKS
[Ø83] 290 RECHTS=RECHTS
[176] 300 REFERENZ=ZAHLEN((LINKS+RECHTS)/2)
[BC4] 310 WHILE LINKS1<=RECHTS1
[5Ø6] 320 WHILE ZAHLEN(LINKS1)<REFERENZ
[GL3] 330 LINKS1=LINKS1+1
[BP1] 340 WEND
[P66] 350 WHILE REFERENZ<ZAHLEN(RECHTS1)
[CW3] 360 RECHTS1=RECHTS1-1
[MP1] 370 WEND
[935] 380 IF LINKS1>RECHTS1 THEN 440
[G84] 390 HILFE=ZAHLEN(LINKS1)
[Z96] 400 ZAHLEN(LINKS1)=ZAHLEN(RECHTS1)
[GP4] 410 ZAHLEN(RECHTS1)=HILFE
[1F3] 420 LINKS1=LINKS1+1
[3X3] 430 RECHTS1=RECHTS1-1
[XG1] 440 WEND
[3U4] 450 IF LINKS1>=RECHTS1 THEN 490
[KF3] 460 LISTEN=LISTEN+1
[394] 470 ANFANG(LISTEN)=LINKS1
[V84] 480 ENDE(LISTEN)=RECHTS
[173] 490 RECHTS=RECHTS1
[3C1] 500 WEND
[P61] 510 WEND
[6M1] 520 RETURN
[VD4] 530 REM Quicksort iterativ
[Y45] 540 REM Zufallszahlen erzeugen
[YZ7] 550 FOR I=1 TO 1000 : ZAHLEN(I)=INT(RND*1000) : N
EXT I
[A93] 560 PRINT "Sortieren"
[PP1] 570 AS=TIME$
[EUI] 580 GOSUB 180
[4R1] 590 ES=TIME$
[B62] 600 PRINT AS,ES
[331] 610 END
```

Listing 3. »quicki.bas« stellt die iterative Quicksort-Variante dar

```
[2N4] 1 REM Programmname: SIER.BAS
[EK4] 2 REM (C) 1988 Everts & Hagedorn
[ZD4] 3 REM DOS International
[L45] 100 REM *****
[L36] 110 REM * S I E R P I N S K I K U R V E N *
[T45] 120 REM *****
[GØ7] 130 REM Eigener Stack für Prozeduraufrufe
[GH2] 140 DIM STACKØ(20)
[F36] 150 REM Einschalten des Grafikmodus
[6S1] 160 SCREEN 2
[SUI] 170 KEY OFF
[Ø51] 180 CLS
[CØ5] 190 GOTO 1470 : ' Hauptprogramm
[5U3] 200 REM *****
[VE2] 210 REM P L O T
[BU3] 220 REM *****
[C95] 230 REM Zeichnen der Grundfigur
[L83] 240 LINE(Y,X)-(YØ,XØ),1
[VX4] 250 LINE(Y+HØ,X+HØ)-(YØ+HØ,XØ+HØ),1
[L24] 260 LINE(Y,X+HØ)-(YØ,XØ+HØ),1
[Z44] 270 LINE(Y+HØ,X)-(YØ+HØ,XØ),1
[S31] 280 XØ=X
[Y31] 290 YØ=Y
[W11] 300 RETURN
[AU3] 310 REM *****
[DY3] 320 REM P R O Z E D U R A
[GU3] 330 REM *****
[ZP4] 340 IF STACKØ(STACKT)<1 THEN 590
[ZU4] 350 STACKT=STACKT+1 : ' Push i-1
[4G5] 360 STACKØ(STACKT)=STACKØ(STACKT-1)-1
[ZM4] 370 GOSUB 310 : ' Prozedur A
[W54] 380 STACKT=STACKT-1 : ' Pop
[LB1] 390 X=X+H
[ZB1] 400 Y=Y+H
[Z3F] 410 GOSUB 200 : REM Plot
[ØU4] 420 STACKT=STACKT+1 : ' push i-1
[XF5] 430 STACKØ(STACKT)=STACKØ(STACKT-1)-1
```



```
[CM4] 440 GOSUB 600      : ' Prozedur B
[N54] 450 STACKT=STACKT-1 : ' Pop
[YP1] 460 X=X+2*H
[LN3] 470 GOSUB 200      : ' Plot
[KU4] 480 STACKT=STACKT+1 : ' Push i-1
[FG5] 490 STACK%(STACKT)=STACK%(STACKT-1)-1
[HN4] 500 GOSUB 1180     : ' Prozedur D
[B54] 510 STACKT=STACKT-1 : ' Pop
[2B1] 520 X=X+H
[BB1] 530 Y=Y+H
[CN3] 540 GOSUB 200      : ' Plot
[BU4] 550 STACKT=STACKT+1 : ' Push i-1
[8G5] 560 STACK%(STACKT)=STACK%(STACKT-1)-1
[BM4] 570 GOSUB 310      : ' Prozedur A
[064] 580 STACKT=STACKT-1 : ' Pop
[VM1] 590 RETURN
[DU3] 600 REM *****
[KY3] 610 REM  P R O Z E D U R  B
[LU3] 620 REM *****
[AU4] 630 IF STACK%(STACKT)<=0 THEN 880
[T23] 640 STACKT=STACKT+1
[7G5] 650 STACK%(STACKT)=STACK%(STACKT-1)-1
[PM4] 660 GOSUB 600      : ' Prozedur B
[833] 670 STACKT=STACKT-1
[SB1] 680 X=X-H
[1C1] 690 Y=Y-H
[4N3] 700 GOSUB 200      : ' Plot
[K23] 710 STACKT=STACKT+1
[0G5] 720 STACK%(STACKT)=STACK%(STACKT-1)-1
[1N4] 730 GOSUB 890      : ' Prozedur C
[133] 740 STACKT=STACKT-1
[9R1] 750 Y=Y-2*H
[PN3] 760 GOSUB 200      : ' Plot
[433] 770 STACKT=STACKT+1
[KG5] 780 STACK%(STACKT)=STACK%(STACKT-1)-1
[NM4] 790 GOSUB 310      : ' Prozedur A
[R23] 800 STACKT=STACKT-1
[5B1] 810 X=X+H
[GB1] 820 Y=Y-H
[FN3] 830 GOSUB 200      : ' Plot
[X23] 840 STACKT=STACKT+1
[BG5] 850 STACK%(STACKT)=STACK%(STACKT-1)-1
[UM4] 860 GOSUB 600      : ' Prozedur B
[C33] 870 STACKT=STACKT-1
[YM1] 880 RETURN
[CV3] 890 REM *****
[RY3] 900 REM  P R O Z E D U R  C
[PU3] 910 REM *****
[D25] 920 IF STACK%(STACKT)<=0 THEN 1170
[W23] 930 STACKT=STACKT+1
[AG5] 940 STACK%(STACKT)=STACK%(STACKT-1)-1
[BN4] 950 GOSUB 890      : ' Prozedur C
[B33] 960 STACKT=STACKT-1
[VB1] 970 X=X-H
[2C1] 980 Y=Y+H
[3P3] 990 GOSUB 200      : ' Plot
[S73] 1000 STACKT=STACKT+1
[VP5] 1010 STACK%(STACKT)=STACK%(STACKT-1)-1
[VP4] 1020 GOSUB 1180     : ' Prozedur D
[CB3] 1030 STACKT=STACKT-1
[6L1] 1040 X=X-2*H
[UN3] 1050 GOSUB 200      : ' Plot
[GB3] 1060 STACKT=STACKT+1
[LR5] 1070 STACK%(STACKT)=STACK%(STACKT-1)-1
[WM4] 1080 GOSUB 600      : ' Prozedur B
[493] 1090 STACKT=STACKT-1
[BD1] 1100 X=X-H
[LD1] 1110 Y=Y-H
[HN3] 1120 GOSUB 200      : ' Plot
[783] 1130 STACKT=STACKT+1
```

```
[AR5] 1140 STACK%(STACKT)=STACK%(STACKT-1)-1
[8N4] 1150 GOSUB 890      : ' Prozedur C
[V83] 1160 STACKT=STACKT-1
[RV1] 1170 RETURN
[UY3] 1180 REM *****
[BR3] 1190 REM  P R O Z E D U R  D
[XX3] 1200 REM *****
[455] 1210 IF STACK%(STACKT)<=0 THEN 1460
[683] 1220 STACKT=STACKT+1
[9R5] 1230 STACK%(STACKT)=STACK%(STACKT-1)-1
[9R4] 1240 GOSUB 1180     : ' Prozedur D
[U83] 1250 STACKT=STACKT-1
[2E1] 1260 X=X+H
[AE1] 1270 Y=Y+H
[CP3] 1280 GOSUB 200      : ' Plot
[293] 1290 STACKT=STACKT+1
[0R5] 1300 STACK%(STACKT)=STACK%(STACKT-1)-1
[7M4] 1310 GOSUB 310      : ' Prozedur A
[H83] 1320 STACKT=STACKT-1
[BL1] 1330 Y=Y+2*H
[2N3] 1340 GOSUB 200      : ' Plot
[N83] 1350 STACKT=STACKT+1
[SR5] 1360 STACK%(STACKT)=STACK%(STACKT-1)-1
[PN4] 1370 GOSUB 890      : ' Prozedur C
[993] 1380 STACKT=STACKT-1
[NE1] 1390 X=X-H
[MD1] 1400 Y=Y+H
[PN3] 1410 GOSUB 200      : ' Plot
[C83] 1420 STACKT=STACKT+1
[FR5] 1430 STACK%(STACKT)=STACK%(STACKT-1)-1
[FR4] 1440 GOSUB 1180     : ' Prozedur D
[093] 1450 STACKT=STACKT-1
[WW1] 1460 RETURN
[ZL4] 1470 REM *****
[AE4] 1480 REM  H a u p t p r o g r a m m
[7M4] 1490 REM *****
[VL2] 1500 X0=230 : Y0=200
[AC3] 1510 H0=256 : H=INT(H0/2)
[C62] 1520 WHILE I<6
[SC1] 1530 I=I+1
[KU1] 1540 X0=X0-H
[P52] 1550 H=INT(H/2)
[VU1] 1560 Y0=Y0+H
[7E1] 1570 X=X0
[FE1] 1580 Y=Y0
[X73] 1590 STACKT=STACKT+1
[KG3] 1600 STACK%(STACKT)=I-1
[SM4] 1610 GOSUB 310      : ' Prozedur A
[AF1] 1620 X=X+H
[UF1] 1630 Y=Y-H
[CP3] 1640 GOSUB 200      : ' Plot
[EN4] 1650 GOSUB 600      : ' Prozedur B
[2G1] 1660 X=X-H
[AG1] 1670 Y=Y-H
[WP3] 1680 GOSUB 200      : ' Plot
[9R4] 1690 GOSUB 890      : ' Prozedur C
[DF1] 1700 X=X-H
[DF1] 1710 Y=Y+H
[7P3] 1720 GOSUB 200      : ' Plot
[8P4] 1730 GOSUB 1180     : ' Prozedur D
[NF1] 1740 X=X+H
[XF1] 1750 Y=Y+H
[RP3] 1760 GOSUB 200      : ' Plot
[2A1] 1770 WEND
[UT4] 1780 AS=INKEYS : IF AS="" THEN 1780
[3Y1] 1790 SCREEN 0
[541] 1800 END
```

Listing 4. »sier.bas« ist eine rekursive Lösung zum Zeichnen von Sierpinski-Kurven

DMV informiert

Wir möchten Sie darauf hinweisen, daß sich unsere Telefonnummer geändert hat!

Die Telefonzentrale erreichen Sie unter:.....0 56 51/80 09-0

Telefax: 0 56 51/80 09-33

Telex:.....9 93 210 dmv d

DMV Verlag · Postfach 2 50 · 3440 Eschwege



PC 1640 Monochrom-Monitor	Amstrad LQ 3500 Drucker	DM 699,-
- mit 1 Laufwerk	DM 1550,-	Amstrad LQ 5000 Drucker
- mit 2 Laufwerken	DM 1669,-	DM 1199,-
- mit 1 Disk + 20 MB HD	DM 2199,-	Amstrad PPC-512 Portable
		DM 1399,-
		weitere Produkte auf Anfrage

PC Tools V4.3 deutsch	DM 109,-
Copy II Option Board Deluxe	DM 249,-
Utah Cobol V5/Fortran	je DM 99,-
JPI-Modula V1.14	DM 179,-

Kostenlose Prospekte auch für Amiga und ST gibt's bei...



CWTC Joachim Tiede
Bergstr. 13, 7109 Roigheim
Tel./BTX 0 62 98/30 98 v. 17-19 Uhr

Unterbrechen Sie mich bitte!

Immer dann, wenn der Computer auf ein bestimmtes Ereignis sofort reagieren soll, muß der Ablauf eines Basic-Programms zwischen- durch unterbrochen werden. GW-Basic stellt sich der Herausforderung mit Befehlen zur situationsbezogenen Ereigniserfassung. Lernen Sie, die Tücken ihrer Anwendung zu umgehen.

Der Cursor blinkt und fordert zu einer Eingabe auf, während in der rechten oberen Ecke die Zeit hochgezählt wird und im Hintergrund ein fröhliches Liedchen ertönt. Solche und ähnliche Ablauf-Sequenzen sind gerade in Basic komfortabel zu programmieren. Ist der Abfang-Mechanismus für ein Ereignis angeschaltet, prüft GW-Basic nach jedem Befehl, ob das erfragte Ereignis schon eingetreten ist, und verzweigt in diesem Fall zu der vereinbarten Zeile. In dieser Zeile beginnt eine Routine, die auf das Ereignis reagiert. Das Unterprogramm endet mit »return«, wonach der Rechner das Programm an der unterbrochenen Stelle fortführt.

Vergegenwärtigen Sie sich, daß nach jeder Anweisung im Programm eine solche Unterbrechung erfolgen kann. So darf in der Ereignisroutine nichts geschehen, was den normalen Programmablauf verwirren könnte. Die Cursor-Position und die augenblickliche Schriftfarbe sind zwischenspeichern, wenn sie im Unterpro-

gramm geändert werden. Wenn Sie überprüfen, ob der Wert einer Variablen innerhalb des Geltungsbereichs einer bestimmten Funktion liegt, und wenn Sie für diese Abfrage zu einem Programmteil verzweigen, in dem diese Funktion mit der Variablen verwendet wird, kann es passieren, daß genau nach der Abfrage das Unterbrechungereignis auftritt. Danach bekommt die Variable möglicherweise einen neuen Wert. Nach der Unterbrechung will der Rechner dann die Funktion ausführen. Wenn aber die Variable mittlerweile über den gültigen Wertebereich hinausgewachsen ist, geschieht ein Ablauffehler. Das Hauptprogramm muß darauf gefaßt sein, daß jederzeit eine Unterbrechung passieren kann. Nötigenfalls muß es die Ereigniserfassung für einen Moment deaktivieren.

Man bedient sich dazu spezieller Steuerkommandos, deren Funktionen auf die zu erfassenden Ereignisse zugeschnitten sind. Die grundsätzliche Syntax lautet:

```
on <Ereignis> (<Wert>) gosub  
<Zeile>
```

Damit ist die Zeilennummer für ein bestimmtes Ereignis festgelegt. Nach der Anweisung

```
<Ereignis> on
```

ist die Ereigniserfassung für das angegebene Ereignis aktiv und nach

```
<Ereignis> off
```

wieder inaktiv. Basic kennt noch eine Form, um die Erfassung eines Ereignisses vorübergehend zu deaktivieren:

```
<Ereignis> stop
```

Tritt nach dieser Anweisung das Ereignis auf, dann speichert der Rechner es vorerst ab, begibt sich aber erst zur vereinbarten Zeile, wenn der Befehl

```
<Ereignis> on
```

die Ereigniserfassung wieder zuläßt, dann jedoch sofort. Diese Stop-Situation ist auch gegeben, wenn sich der Rechner gerade im Unterbrechungsprogramm befindet. Ist die Routine durch »return« beendet, springt sie der Computer gleich noch einmal an, wenn zwischenzeitlich das Ereignis wieder eingetreten ist. Im Bild sehen Sie, wie diese Zustände auf die Ereigniserfassung Einfluß nehmen. Die wichtigste und wohl am häufigsten verwendete Funktion aus der Ereigniserfassung hält sich nicht an diesen Standard. Sie dient dazu, eventuell auftretende Ablauffehler abzufangen und auf sie zu reagieren. Greift der Rechner im Programm auf seine Peripherie zu, kann es immer passieren, daß dabei aus irgendeinem Grund ein Fehler auftritt, der dann einen Programmabbruch zur Folge

Fehlernummer	Bedeutung
1	»Next« ohne »For«
2	Syntax-Fehler
3	»Return« ohne »Gosub«
4	Zu wenige Data-Elemente
5	Ungültiger Funktionsaufruf
6	Überlauffehler
7	Speicher reicht nicht aus
8	Nichtdefinierte Zeilennummer
9	Ungültige Feld-Indizierung
10	Wiederholte Feld-Dimensionierung
11	Division durch Null
12	Unzulässiger Direktaufruf
13	Falscher Variablentyp
14	String-Speicher reicht nicht aus
15	Zeichenkette ist zu lang
16	String-Operation ist zu kompliziert
17	Anweisung »Cont« ist nicht ausführbar
18	Nicht definierte Funktion
19	Resume-Anweisung erwartet
20	»Resume« ohne Fehler
22	Operand erwartet
23	Zeilenpuffer überlaufen
24	Gerät antwortet nicht
25	Falsches Gerät
26	»For« ohne »Next«
27	Kein Papier mehr im Drucker
29	»While« ohne »Wend«
30	»Wend« ohne »While«

Tabelle 1. Die Programm- und Syntaxfehlernummern von GW-Basic

Fehlernummer	Bedeutung
50	Field-Vereinbarung größer als Satzlänge
51	Interner Interpreter-Fehler
52	Ungültige Dateinummer
53	Datei nicht gefunden
54	Falscher Dateimodus
55	Datei bereits geöffnet
57	Ein-/Ausgabefehler am Gerät
58	Datei existiert bereits
61	Diskette voll
62	Eingabe nach Dateieinde
63	Ungültige Satznummer
64	Falscher Dateiname
66	Direktanweisung in Eingabedatei
67	Zu viele Dateien im Verzeichnis
68	Gerät ist nicht vorhanden
69	Überlauf des Com-Eingabepuffers
70	Diskette ist schreibgeschützt
71	Laufwerk ist nicht bereit
72	Fataler Diskettenfehler
73	Reserviertes Wort
74	Umbenennung über Laufwerke
75	Pfad- oder Dateizuordnung unmöglich
76	Pfad nicht gefunden
77	Verklemmung (Deadlock)

Tabelle 2. GW-Basics Ein-/Ausgabefehlernummer

hat. Bekam der Rechner zuvor aber die Anweisung

```
on error goto <Zeilennummer>
```

erfolgt im Fehlerfall die weitere Bearbeitung ab der angegebenen Zeile. Dort sollte ein Unterprogramm beginnen, das die Fehlerursache feststellt und den Benutzer zu einer Korrektur auffordert – sofern das noch möglich ist. Im einfachsten Fall besteht es aus der Zeile

```
fehler=err:resume next
```

wonach die weitere Fehlerbearbeitung dem Hauptprogramm überlassen wird, das am Inhalt der Variable »fehler« erkennen kann, ob, und wenn ja, welcher Fehler aufgetreten ist.

»on error goto« – Programmfehler abfangen, analysieren und bearbeiten

Typische Fehlersituationen, die sich so bearbeiten lassen, liegen im Öffnen von Dateien, deren Bezeichnung der Benutzer wählt. Die Dateibezeichnung ist unzulässig, eine zum Lesen geöffnete Datei existiert nicht oder ein bezeichnetes Gerät ist nicht bereit oder nicht vorhanden. Der Rechner speichert zunächst die Nummer des Fehlers in der Systemvariablen »err«, und die Zeilennummer, in der der Fehler auftrat in »erl«. Die Tabellen 1 und 2 zeigen die übergebenen Fehlernummern und deren Bedeutung. Diese Fehler lassen sich mit dem Kommando »error <Fehlernummer>« auch selbst erzeugen. Ist die Fehlerbehandlung aktiv, verursachen Sie so einen bewußten Sprung zur Fehlerroutine, der Sie mit der Fehlernummer sogar einen Wert übergeben

können. Die Fehlernummern 21, 28, 31 bis 49, 56, 59, 60, 65 und 78 bis 255 sind nicht belegt. Basic bezeichnet sie als »unbekannte Fehler«. Verwenden Sie sie, um eigene Fehler zu spezifizieren, und behandeln Sie solche Fehler in der Fehlerroutine. Für die Fans der Bios-nahen Programmierung ist die Variable »error«: Sie enthält die »kritische Fehler«-Nummer im Falle eines Gerätefehlers gemäß Interrupt 24h. Das im Detail zu erklären, würde aber zu weit führen.

Die Rückkehr aus der Fehlerroutine kann auf dreierlei Wegen geschehen. Das Schlüsselwort »resume« führt den Rechner an die fehlerverursachende Anweisung zurück. Er führt diese also noch einmal aus, was bedeutet, daß die Ursache des Fehlers nicht in der Anweisung selbst lag und die Fehlerroutine ihn beheben konnte. Denkbar wäre, daß dort so lange verblieben wurde, bis der Drucker betriebsbereit war. Nach »resume next« setzt der Computer die Arbeit nach dem fehlerhaften Befehl fort und nach »resume <Zeilennummer>« an der angegebenen Zeile. Als programmglobale Einrichtung empfiehlt sich die On-Error-Anweisung nicht. Erstens kommt es höchst selten vor, daß alle im Programm auftretenden Fehler die gleiche Behandlung verdienen, und zweitens stiftet ein logischer oder syntaktischer Programmfehler dann echte Verwirrung. Das geht soweit, daß der Interpreter nach einem Eingabefehler im Direktmodus das Fehlerprogramm abzuarbeiten beginnt und dort zum Beispiel auffordert, Papier in den Drucker zu füllen.

Sinnvoll eingesetzt ist das »Error-Trapping« schon, wenn es kurz vor einer fehlerverdächtigen Arbeit aktiviert und danach mit »on error goto 0« wieder deaktiviert wird (Listing 1); die Befehle »error on«, »error off« und »error stop« existieren ausnahmsweise nicht. Listing 2 ist ein Basic-Programm, das nur durch Fehler gesteuert wird. Die tatsächlichen Anweisungen befinden sich in der Fehlerroutine, die anhand der Fehlernummer zu den entsprechenden Anweisungszeilen verzweigt. Diese Methode ist wohl weniger dazu geeignet, übersichtliche Programmierung aufzuzeigen, als vielmehr auf erheiternde Weise die Arbeit der Funktion zu verdeutlichen.

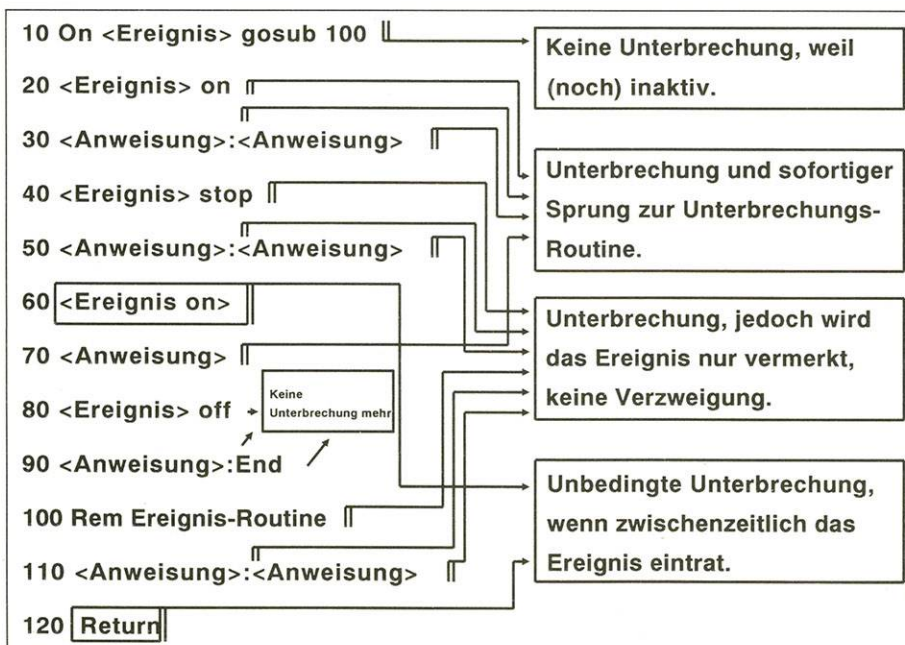
Ein besonderes Merkmal der Fehlerbehandlung ist der Umstand, daß der Rechner auf kein anderes Ereignis reagiert, solange er sich in der Fehler-Routine aufhält. Allerdings vermerkt er sie und verursacht sie nach dem »resume« sofort.

»on timer() gosub« – Unterbrechung in vereinbarten Zeitintervallen

Soll der Rechner stündlich oder minutlich eine Arbeit ausführen, egal was er im Hauptprogramm zu tun hat, bedient man sich der On-Timer-Anweisung.

```
on timer(x) gosub <Zeile>
```

Nachdem mit dem Timer-On-Befehl die Ereigniserfassung aktiviert ist, ruft der Rechner alle x Sekunden das Unterprogramm ab »Zeile« auf. Das Unterprogramm muß mit »Return« abgeschlossen sein. Je häufiger der Rechner das Unterprogramm aufruft, je kleiner also ist x, desto langsamer arbeitet das Hauptprogramm – denn es wird häufiger unterbrochen (höchstens sekundlich bei x = 1). Zumeist realisiert ein solches Unterprogramm die Zeitanzeige in einer Bildschirm-ecke. Speichert man die momentane Position des Cursors und die Zeichenfarbe zwischen und setzt sie am Ende der Routine wieder auf diese Werte, kann eigentlich nichts mehr passieren. Diese Annahme ist aber falsch. Befindet sich der Cursor in der rechten unteren Bildschirm-ecke und wurde gerade das letzte Zeichen der Zeile ausgegeben, bleibt der Cursor vorerst dort und schiebt erst vor Ausgabe des nächsten Zeichens den Bildschirm um eine Zeile nach oben (»Scrolling«). Bekommt der Cursor aber, wenn auch nur vorübergehend, nach dem letzten Zeichen eine neue Position, dann hat der Rechner vergessen, daß er die letzte Spalte in der letzten Zeile schon beschrieben hat und setzt das nächste Zeichen über das zuvor ausgedruckte, scrollt also



So arbeitet GW-Basic mit den Steuerbefehlen der Ereigniserfassung, wenn an der unterbrochenen Stelle das Ereignis eintrat

nicht. Deshalb sind alle Bildschirm-
ausgaben in der Timeroutine über
das direkte Beschreiben des Bild-
schirmspeichers vorzunehmen. Besit-
zen Sie eine Hercules-Grafikkarte
oder eine reine Textkarte, müssen Sie
das Datensegment mit »def
seg = &Hb000« an diese Adresse set-
zen. Alle anderen Grafikkarten ken-
nen den Bildschirm an der Adresse
B800hex. Mit dem Poke-Befehl kön-
nen Sie den Bildschirm dann direkt
beschreiben, wobei die linke obere
Ecke den Wert 0 hat, die Stelle dane-
ben den Wert 2 und die darunter den
Wert 160. Die Bildschirmpositionen
sind immer zwei Speicherwerte von-
einander entfernt, weil sich dazwi-
schen stets die Farb- oder Attribut-
werte der zugehörigen Positionen be-
finden. Haben Sie die Zeit auf dem
Bildschirm kontinuierlich darzustel-
len, lesen Sie mit der Asc-Funktion
die Werte aus der Variablen »time\$«
und »poken« sie an die entsprechen-
den Stellen am Schirm. So entstehen
keine Probleme mit der Bildschirm-
ausgabe. Ist es einmal absolut ausge-
schlossen, daß der Bildschirm in ei-
nem Programm scrollt, dürfen Sie auf
die normale Bildschirmausgabe mit-
tels »print« zurückgreifen. Vergessen
Sie aber nicht, die augenblickliche
Schriftfarbe in einer Variablen abzu-
legen, wenn Sie sie in der Timerou-
tine ändern. Im Gegensatz zur Cur-
sor-Position läßt sie sich nicht mit ei-
ner Funktion erfragen. In Listing 3
sehen Sie ein Spiel (»Master Mint«),
das dem Spieler für jeden Zug 80 Se-
kunden Zeit läßt, eine vom Rechner
erdachte vierstellige Zahl zu erarbei-
ten. Die vergangene Zeit stellt das
Programm mit Hilfe der Timerfunk-
tion als Anzeigebalken in der letzten
Zeile dar.

»on key() gosub« – Von der schnellen Reaktion auf Ta- statureingaben

Die wohl heikelste Funktion aus der
Palette der Ereigniserfassungen von
GW-Basic arbeitet mit der Tastatur.
Standardmäßig kann man die
Funktions- und die Cursor-Tasten ab-
fangen, für die Kontrolle jeder Taste
muß aber gesondert Anweisung gege-
ben werden. Nach dem Befehl

```
on key(n) gosub <Zeile>
```

weiß der Rechner, wohin er verzwei-
gen soll, wenn der Benutzer die Taste
mit der Nummer n (die Werte entneh-
men Sie der Tabelle 3) betätigt hat.
Der Befehl »key(n) on« aktiviert den
Abfangmechanismus für die bezeich-
nete Taste. Die Problematik liegt in
der Natur der Sache: Üblicherweise
muß man auf mehrere Tasten reagie-

Wert	Taste
1 bis 10	[F1] bis [F10]
11	[Cursor oben]
12	[Cursor rechts]
13	[Cursor links]
14	[Cursor unten]
15 bis 20	Definierte Tasten (nur bei Basica)

**Tabelle 3. Die Nummern der mit »on
key() gosub« standardmäßig abfangbaren
Tasten**

ren, beispielsweise auf den ganzen
Cursor-Block oder auf eine Reihe von
Funktionstasten. Erforderlichenfalls
ist also jede Taste auch gesondert zu
deaktivieren – während dieses Vor-
gangs kann es passieren, daß der An-
wender unerwünschterweise eine ge-
rade noch aktive Taste betätigt. Je
mehr Tasten abzufangen sind, desto
mehr Rechenzeit braucht der Rech-
ner, um die Tasten an- oder auszu-
schalten. Man kann sich diesen Lu-
xus also nicht unendlich häufig gön-
nen, auch dann nicht, wenn die Pro-
grammstruktur es erfordert. Am
besten geht man dazu über, die Un-
terbrechungsroutine diesen Proble-
men anzupassen. Nachdem das Pro-
gramm alle benötigten Tasten mit

```
on key(n) on
```

aktiviert hat, schaltet es sie erst zum
Programmende oder zu einem Zeit-
punkt, zu dem sie endgültig nutzlos
wurden, mit

```
on key(n) off
```

wieder ab. In einer Kontrollvariablen
steht dann, wie wünschenswert die
Reaktion auf ein Tastaturereignis ge-
rade ist. Zum Beispiel:

```
0:erwünscht  
1:unerwünscht  
2:momentan unerwünscht
```

Im letzten Fall kann die Routine er-
kennen, ob das Ereignis in einer an-
deren Variablen festzuhalten ist und
das Hauptprogramm zu einem späte-
ren Zeitpunkt darauf reagieren will,
indem es vielleicht die Tastaturrou-
tine gesondert »besucht«. Wie das alles
in der Praxis aussieht, verrät ein
Blick auf Listing 4. Die Kontrollvaria-
ble ist hier »a«. Ab der Zeile 1000 be-
ginnt die Unterbrechungsroutine.
Hier bekommt die Variable »spzug«
in den ersten vier relevanten Zeilen
einen Kennwert für die betätigte Ta-
ste. Erst in der Zeile 1050 überprüft
die Routine, ob, und wenn ja wie, sie
auf den Tastendruck zu reagieren
hat. Besitzt »a« den Wert 2, ist die
Rückkehr zum Hauptprogramm be-
schlossen, wobei »spzug« ihren Wert
behält. Gilt »a« > 0, bekommt
»spzug« den Wert 0 und die Routine
ist ebenfalls beendet. In jedem ande-
rem Fall bewegt der Rechner die

Scancode	Taste
0	Caps- und Num-Lock nicht aktiv
1	Rechte Shift-Taste
2	Linke Shift-Taste
4	Ctrl-Taste
8	Alt-Taste
16	Num-Lock aktiv
32	Caps-Lock aktiv

**Tabelle 4. Unter Basica kann man mit
diesem Tastatur-Scancode verknüpfte
Tasten abfangen**

Spielfigur in die gewünschte Rich-
tung, sofern sie damit nicht über den
Bildschirmrand hinausgeführt wird.

Das Hauptprogramm läßt nur dann
einen Spielerzug zu, wenn es sich ge-
rade in der Warteschleife befindet,
welche die einzufangende Figur ver-
langsamt. An allen anderen Stellen
im Programm kann eine Tastaturun-
terbrechung stören. Kurz vor der
Schleife »besucht« der Rechner die
Routine für den Spielerzug explizit,
wenn »spzug« größer ist als 0, also zu
einem ungünstigen Zeitpunkt eine
Cursor-Taste gedrückt wurde. Sobald
die Figur (»Willi«) eingefangen ist,
steht in der Variablen »a« eine 1. Ab
diesem Zeitpunkt darf die Tastatur
keine Steuerung mehr vornehmen,
das Spiel ist beendet.

Weil das Spiel ohne Farben aus-
kommt, muß auf diese auch keine
Rücksicht genommen werden. An-
dernfalls ist aber genauso vorzuge-
hen, wie bei der Zeitunterbrechung:
die Farbe ist in einer Variable festzu-
halten.

Ist es einmal nicht notwendig, daß
das Programm auf eine Taste sofort
reagiert, sollte man im Unterbre-
chungsprogramm lediglich vermer-
ken, daß die Taste gedrückt wurde
und dem Hauptprogramm alle Steue-
rungsaufgaben überlassen. Das ver-
ringert das Risiko eines irrlaufenden
Programms am effektivsten.

Wer mit einer Version von »Basica«
arbeitet, darf dessen zusätzliche Op-
tion nutzen, selbstdefinierte Tasten-
kombinationen abzufangen. Dazu de-
finiert man eine imaginäre Funk-
tionstaste (15 bis 20) mit dem Charac-
ter-Wert des sogenannten
Keyboard-Flags, einer Kennung für
den Zustand der Shift-, Ctrl-, Alt-,
Num-Lock- und Caps-Lock-Tasten
(Tabelle 4) und einer zusätzlich ged-
rückten Taste. Für das Abfangen der
Kombination [Shift-Esc] sind also fol-
gende Programmschritte vorzu-
nehmen:

```
key 15,chr$(2)+chr$(27)  
on key(15) gosub <Zeile>  
key(15) on
```

Jedesmal wenn der Anwender diese
Kombination betätigt, verzweigt der

Rechner an das Unterprogramm ab
<Zeile>.

Die Datenfernübertragung (DFÜ) ist eine auf Übertragungszeit-Optimierung ausgelegte Arbeit. Der ohnehin schon verhältnismäßig langsame Vorgang darf nicht auch noch zusätzlich dadurch behindert werden, daß das aktuelle Basic-Programm eben gerade nicht an der Zeile abgearbeitet wird, an der sich die Anweisungen zur Schnittstellenbetreuung befinden. Sowie ein Zeichen zur Eingabe an der Schnittstelle anliegt, muß das Programm es holen, egal, was es gerade tut. Andernfalls muß der Rechner den Übertragungspartner anweisen, zu warten, was er ganz automatisch macht. Das bedeutet aber unnötig häufige Verzögerungen. Der Befehl

```
on com(n) gosub <Zeile>
```

vereinbart die weitere Bearbeitung ab <Zeile>, sobald an der seriellen Schnittstelle ein Zeichen anliegt. »n« ist die Nummer der Schnittstelle, also 1 für »com1« und 2 für »com2«.

»on com() gosub« – Das Problem mit der seriellen Schnittstellenkommunikation

Listing 5 ist ein kleines Terminalprogramm, das mit der On-Com-Anweisung arbeitet. Zuerst öffnet man die Schnittstelle mit dem eigens dafür vorgesehenen Open-Com-Befehl:

```
open "com<n>":<Baud-Rate>,<Parität>,<Daten-Bits>,<Stop-Bits>,<Parameter,...>" as <Kanalnummer>
```

Diese Begriffe verwirren auf Anhieb vielleicht, sind aber reine Vereinbarungsangelegenheit, das heißt, beide Übertragungsstellen müssen mit den gleichen Parametern kommunizieren, sonst geht es nicht. Die »Baud-Rate« ist die Geschwindigkeit in Bit pro Sekunde. Mit einem Akustik-Koppler überträgt man meistens mit 300 Baud, mit einem Telefon-Modem 1200 bis 2400 Baud und direkt, also mit einem Verbindungskabel von einem Rechner zum anderem unter Basic, mit höchstens 9600 Baud. Der wichtigste Befehlsparameter des Open-Com-Befehls ist »pe«, wodurch man verhindert, daß Basic bei Übertragungsfehlern das Programm mit einem Fehler beendet.

Nach dem üblichen Aktivieren »com(n) on« wechselt Basic zu der zuvor vereinbarten Zeile, sobald sich ein Zeichen zur Eingabe an der Schnittstelle meldet. Dort können auch schon mehrere Zeichen anliegen. Wie viele das sind, ergibt die Rechnung »<Puffergröße>-lof(<Kanalnummer>)«. Im Gegensatz zur normalen Dateiverarbeitung erhält man von der Lof-Funktion diesmal

nicht die Dateilänge, sondern wie viele Zeichen noch im Ein-/Ausgabepuffer der Schnittstelle frei sind. Subtrahiert man diesen Wert von der Puffergröße (normalerweise 256 Byte, mit der Option /c beim Aufruf von GW-Basic läßt sich das aber selbst bestimmen), weiß man, wie viele Zeichen »abzuholen« sind. Mit der Input\$-Funktion lesen Sie dann genau diese Anzahl von Zeichen ein. Soll das Ergebnis auf dem Bildschirm dargestellt werden, muß beachtet werden, daß Basic nicht alle Zeichen so darstellt, wie es für die Datenfernübertragung nötig ist. Das ASCII-Zeichen 13 (CR) ist nur ein Wagenrücklauf, setzt den Cursor also an die erste Spalte der Zeile. Erst das Zeichen 10 (LF) bewegt den Cursor um eine Zeile nach unten. Das Zeichen 8 (BS) ist unter Basic ein invertierter Kreis, soll aber den Cursor um eine Stelle nach links bewegen und das Zeichen dort löschen. Mit einer Schleife prüfen Sie am besten jedes Zeichen des gelesenen Strings und simulieren die richtige Funktion dieser – von Basic mißverstandenen – Steuerzeichen. Solange die Schnittstellenroutine, wie in Listing 5 der einzige Programmteil ist, in dem Bildschirmausgaben geschehen, müssen keine gesonderten Maßnahmen getroffen werden. Andernfalls ist aber mit den bereits genannten Vorkehrungen zu arbeiten.

Musik, Musik, Musik – Mit »on play() gosub« realisieren Sie beliebig lange Abspielsequenzen

Der Play-Befehl ist ein beliebtes Mittel, einen tristen Programmablauf mit musikalischen Einlagen aufzulockern. Allerdings ist die Kapazität des Play-Puffers beschränkt. Sobald längere Stücke zum besten gegeben werden sollen, muß der Rechner warten, bis alle noch zu spielenden Noten in seinem Puffer Platz haben. Man bedient sich hier also wieder eines Unterbrechungsbefehls.

```
on play(n) gosub <Zeile>
```

Nach diesem Kommando weiß der Rechner, daß er unverzüglich das Unterprogramm ab »Zeile« aufzusuchen hat, sobald die letzte Play-Anweisung nur noch n Noten spielen muß. Das passiert natürlich erst, sobald die Ereigniserfassung durch

```
play on
```

aktiv ist. Das ganze Musikstück muß in kleinere Stücke unterteilt sein, für die gilt: sie müssen genau halb so lang sein, wie der Rechner sie gerade noch ohne Wartezeit abspielen könnte. Der ersten Play-Anweisung folgt die Vereinbarung, wo der nächste

Play-Befehl mit dem anschließenden Musikteil zu finden ist. Dort geht es nach gleichem Muster weiter, bis das Stück endet.

Entwerfen Sie zunächst das Musikstück. Unterteilen Sie es danach in Sequenzen, die der Rechner einzeln gerade noch im Hintergrund spielen kann, ohne daß er auf Noten warten muß, die nicht mehr in den Puffer passen. Teilen Sie dann jede der Zeichenketten in zwei Teile auf (und zwar genau in der Mitte). Das sind die portionierten Sequenzen für das Unterbrechungsprogramm. Jede Sequenz soll in einem eigenen Play-Befehl abgearbeitet werden. Ihr folgt die Vereinbarung, wo die anschließende Play-Sequenz zu finden ist. Listing 6 spielt im Hintergrund ein zweigeteiltes Musikstück, und zählt im Vordergrund ununterbrochen bis 9999. Dort steht der Play-On-Befehl noch vor der Anweisung, die Zeilennummer und Unterbrechungswert enthält. In der Zeile 20 initialisiert die Play-Anweisung Tonhöhe und -länge sowie den Abspielmodus, in diesem Fall »ml« für zusammenhängendes Abspielen (»ms« stünde für »staccato«) und »mb« für die Hintergrundverarbeitung (im Gegensatz zu »mf« für direkte Bearbeitung). Danach springt das Hauptprogramm zu der Zeile 100, die die erste Sequenz abspielt, die nächste Zeile vereinbart und wieder zurückkehrt. Das ist etwa mit dem Ankurbeln eines Grammophons vergleichbar. Von da ab muß sich das Hauptprogramm um die Musik nämlich nicht mehr kümmern.

In Zeile 110 steht, daß der nächste Play-Befehl in der Zeile 120 relevant ist, sobald nur noch 16 Noten zu spielen sind. Die Zeile 130 führt dann die Musik bei Zeile 100 fort, wenn nur noch 17 Noten im Puffer sind, und so weiter. Es ist zweckmäßig, die Unterbrechungswerte immer möglichst hoch zu halten – sonst kann es passieren, daß die Musikkette abreißt. Der Grund: Der vereinbarte Wert war erreicht, das Hauptprogramm ließ sich aber gerade nicht unterbrechen; es sind zu wenige Noten im Puffer, um die Unterbrechung nachzuholen. Denn hat ein Play-Kommando einmal unverknüpft geendet, findet keine Unterbrechung mehr statt. Der Rechner prüft nur so lange auf die On-Play-Unterbrechung, wie er einen Musikteil abspielt.

Ärgerlich ist es, wenn man bei einem Computerspiel verzweifelt auf den Feuerknopf des Joystick hämmert und das Programm keinen Schuß auslöst, weil ihm das nicht früh genug auffällt. Bei der Programmierung von Action-Spielen muß man darauf Rücksicht nehmen und eine schnelle Reaktion gewährleisten. Das ge-

Wert	Knopf	Port
0	1	A
2	1	B
4	2	A
6	2	B

Tabelle 5. »on strig(n)« fängt die Aktivität zweier Feuerknöpfe (1 oder 2) an zwei Gameports (A und B) ab, wobei n der zugehörige Wert aus der Tabelle ist

schieht am besten wieder mit der Ereigniserfassung, denn Basic gestattet es, dieses Ereignis abzufangen. Der Befehl

```
on strig(n) gosub <Zeile>
```

heißt den Rechner, sobald der Feuerknopf n gedrückt ist, das Unterprogramm ab »Zeile« aufzusuchen. Der Wert für n stellt dabei eine Kennzahl dar, die einen der Feuerknöpfe 1 oder 2 des Joystick in Gameport A oder B spezifiziert (Tabelle 5). Nach

```
strig(n) on
```

prüft der Interpreter nach jeder Basic-Anweisung, ob der bezeichnete Knopf gedrückt ist (oder war) und verzweigt in diesem Fall zur gewünschten Zeile. Dort beginnt wie immer ein Unterprogramm, das das Ereignis behandelt und mit »Return« wieder zurückkehrt. Soll die Betätigung des Joystick-Knopfs einen Schuß auslösen, so ist man gut beraten, diesen nicht in der Routine vollständig auszuführen, weil das Hauptprogramm in dieser Zeit nicht weiterarbeitet. Figuren, die sich bewegen, bleiben solange also stehen. Man speichert die Position der Spielfigur als Schußkoordinate zwischen und gibt einer Kennvariablen einen Wert, der dem Hauptprogramm anzeigt, daß ein Schuß auszulösen ist. Das Programm »asteroid« (Listing 7) geht so vor. Die Erkennung des Feuerknopf-Ereignisses ist für die Zeile 3000 beschlossen, wo das Programm zunächst prüft, ob bereits ein Schuß unterwegs ist. Wenn das Geschöß fliegt, kehrt die Routine zurück. Andernfalls bekommt die Kennvariable »sch« den Wert 1, das Feld für die Schußrichtung den Wert der Standardrichtung des Schießenden, und die Koordinatenvariablen des Geschosses die Werte der Koordinaten der Spielfigur. Danach ist die Routine beendet (»Return«). Die Zeile 240 verursacht einen Sprung zur Schußbearbeitung, sofern »sch« größer ist als »0«. Die Routine erkennt am Wert »1«, daß dieser Schuß gerade erfolgt ist, es also nicht erforderlich ist, die letzte Geschößposition mit einem Leerzeichen zu löschen. Der Schuß bewegt sich ein Zeichen in seine Richtung, und »sch« bekommt den Wert »2« – der Schuß ist nicht mehr neu. Hat der Schuß das Ziel getroffen oder ist er

Wert	Ergebnis
0	Bewegungswert des Joystick A in x-Richtung (Erst mit der Abfrage stick(0) aktualisiert Basic die Ergebnisse für die Abfragen 1, 2 und 3)
1	Bewegungswert des Joystick A in y-Richtung
2	Bewegungswert des Joystick B in x-Richtung
3	Bewegungswert des Joystick B in y-Richtung

Tabelle 6. Die Funktion »stick(n)« übergibt ein Ergebnis, das vom Wert abhängig ist, der für n eingesetzt wurde

am Bildschirmrand angekommen, wird er gelöscht und »sch« wieder auf »0« gesetzt. Dieses Unterprogramm läuft so oft wie möglich ab, in diesem Fall jedesmal, bevor eine der bis zu 20 feindlichen Figuren zieht. Der Schuß ist also 19mal schneller als der der Feinde.

Die restliche Joystick-Steuerung arbeitet mit der Funktion »stick(n)«, wobei n ein Kennwert ist, der die Art der Information, die man von der Funktion will, angibt (Bewegungsrichtung in x- und y-Richtung des Joystick in Gameport A oder B). Tabelle 6 zeigt, welche Ergebnisse man von der Stick-Funktion für alle einsetzbaren Werte erwarten kann.

Ist die Ereigniserfassung für den Feuerknopf nicht notwendig, kann man von der Funktion »strig(n)« auch direkt erfragen, ob der Knopf gedrückt ist oder war. Der Wert n steht wieder

für die Art der gewünschten Information (Tabelle 7).

Besitzen Sie keinen Joystick, laden Sie das Programm »asteroid« und binden sie das Overlay-Programm (Listing 8) mit

```
merge "ovr"
```

ein, und speichern Sie es. Die Steuerung funktioniert dann mit den Cursor-Tasten.

»on pen gosub« – Schnell erfassen, wo der Stift auf dem Bildschirm malt

Einmal ein Boom der Computertechnologie, erweist sich der Lichtgriffel inzwischen nur als mäßig nützliches Instrument. Die Arbeit damit ist ermüdend und ungenau. Konzipiert zur Zusammenarbeit mit möglichst jeder damals bekannten Hard-

Wert	Ergebnis
0	-1 wenn nach der letzten Strig(0)-Funktion der Feuerknopf 1 des Joystick in Gameport A gedrückt wurde, andernfalls 0.
1	-1 wenn der Feuerknopf 1 des Joystick in Gameport A gerade gedrückt ist, andernfalls 0.
2	-1 wenn nach der letzten Strig(2)-Funktion der Feuerknopf 1 des Joystick in Gameport B gedrückt wurde, andernfalls 0.
3	-1 wenn der Feuerknopf 1 des Joystick in Gameport B gerade gedrückt ist, andernfalls 0.
4	-1 wenn nach der letzten Strig(4)-Funktion der Feuerknopf 2 des Joystick in Gameport A gedrückt wurde, andernfalls 0.
5	-1 wenn der Feuerknopf 2 des Joystick in Gameport A gerade gedrückt ist, andernfalls 0.
6	-1 wenn nach der letzten Strig(6)-Funktion der Feuerknopf 2 des Joystick in Gameport B gedrückt wurde, andernfalls 0.
7	-1 wenn der Feuerknopf 2 des Joystick in Gameport B gerade gedrückt ist, andernfalls 0.

Tabelle 7. Den Zustand eines der Feuerknöpfe der Joysticks verrät die Abfrage »strig(n)«, wobei für n die Werte aus der Tabelle gelten

Wert	Ergebnis
0	-1 wenn der Knopf des Lichtgriffels seit der letzten Pen-Funktion betätigt wurde, andernfalls 0.
1	Neueste x-Koordinate des Lichtgriffels gemessen in Bildschirmpunkten
2	Neueste y-Koordinate in Bildschirmpunkten
3	-1 wenn der Knopf des Lichtgriffels gerade gedrückt ist, andernfalls 0.
4	Letzte bekannte (gültige) x-Koordinate des Lichtgriffels in Bildschirmpunkten
5	Letzte bekannte (gültige) y-Koordinate in Bildschirmpunkten
6	Neueste y-Koordinate des Lichtgriffels in der Text-Matrix (80 mal 25)
7	Neueste x-Koordinate in der Text-Matrix
8	Letzte bekannte (gültige) y-Position des Lichtgriffels in der Text-Matrix
9	Letzte bekannte (gültige) x-Position in der Text-Matrix

Tabelle 8. Die Ergebnisse der jeweils für n eingesetzten Werte der Funktion »pen(n)«

MaskEdit PLUS V. 4.1

Professioneller Maskengenerator

für Turbo Pascal 3.0, 4.0, 5.0, dBase III (plus) und C

MASKEDIT PLUS bietet die Möglichkeiten, die Sie bei professioneller Software erwarten:

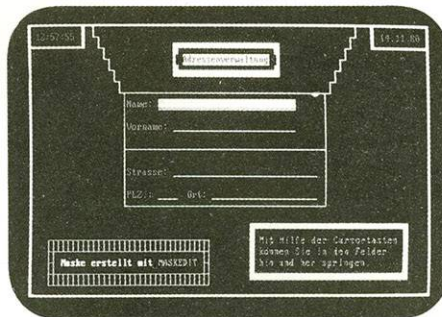
Zaubern Sie Bildschirmmasken mit MASKEDIT PLUS in Sekundenschnelle. Alle Funktionen über Pull-downmenüs oder Funktionstasten aufrufbar. Linien und Rahmen, auch farbig, einfach mit der Maus oder Cursor zu zeichnen. Jederzeit kontextbezogene Hilfe auf Tastendruck.

Technische Details:

universeller Maustreiber (auch für andere Programme nutzbar) · 40 Funktionen zur Maskenerstellung · 8 verschiedene Rahmentypen · Teile der Maske können kopiert, gelöscht, verschoben, zentriert, farbig gestaltet und mit beliebigen Zeichen gefüllt werden · Zeilen einfügen und löschen · der komplette ASCII-Zeichensatz steht zur Verfügung (auch Zeichen kleiner 32) · gleichzeitige Bearbeitung von verschiedenen Masken · Sonderzeichen können auf Funktionstasten gelegt werden · Festlegung von Ein- und Ausgabefeldern · automatisches Anpassen von Kreuzungspunkten bei Linien und Rahmen · Undo-Funktion, um versehentliche Änderungen rückgängig zu machen · Farbeinstellung mit Hilfe einer Farbtabelle · Sonderzeichen jederzeit über eine integrierte ASCII-Tabelle zugänglich · diverse Ausgabeformate: ASCII, Pascal, Pascal-Units, dBase und C

Highlights des Turbo-Pascal-Maskentreibers:

Aufruf generierter Masken durch einen einzigen Befehl! · Steuerung der Eingabe-/Ausgabefelder übernimmt der Maskentreiber · der Benutzer kann sich mit Cursorstasten frei in der Eingabemaske bewegen · Eingabemaske kann mit definiertem Tastendruck (z.B. ESC) verlassen werden · Teilmenüs des Zeichensatzes können für bestimmte Felder als nicht zulässig ausgeklammert werden · der



Treiber kann veranlaßt werden, bei Änderung von bestimmten Feldern die Kontrolle an das Programm zu übergeben, um die Eingabe zu überprüfen · automatische Typüberprüfung der Eingabe durch den Maskentreiber · Datum und Uhrzeit können in die Masken integriert werden

Notwendige Konfiguration:

PC XT/AT mit 256 kByte oder mehr, 1 Diskettenlaufwerk bzw. Festplatte, MS-DOS ab Version 2.11

(unverbindliche Preisempfehlung)

MaskEdit PLUS

99,- DM*

Disk 5 1/4", Best.-Nr. 238 Disk 3 1/2", Best.-Nr. 239

Die BASIC2-Toolbox

Das Buch zu BASIC2

PC 1512/1640-Besitzer können aufatmen. Jetzt ist sie da, die BASIC2-Toolbox. Mit diesem einzigartigen Buch sind Sie nun in der Lage, das Locomotive BASIC2 besser auszunutzen, die Programmierung wird zum Kinderspiel.

Die BASIC2-Toolbox bietet nicht die x-te Einführung, sondern gibt anhand leistungsstarker und praxisorientierter Routinen und Programme den nötigen Durchblick. Schritt für Schritt lernen Sie die strukturierte Programmentwicklung bis hin zum fertigen und lauffähigen Programm. Der Text ist in leicht und für jedermann verständlicher Form geschrieben, der Lerneffekt ist quasi garantiert. Alle Beispielprogramme sind sofort nachvollziehbar und stammen aus der täglichen Anwendungspraxis.

Einige Beispiele aus dem Inhalt:

- Kurze Einführung in die Software-Entwicklung
- Werkzeuge für BASIC2
- Werkzeuge zur Behandlung von Textdateien
- Werkzeuge zur Software-Entwicklung
- Ein kompletter Disassembler für den Intel 8086



Die BASIC2-Toolbox
Autor: Günter Born,
250 Seiten, 54 Abb.,
ISBN Nr. 3-926177-01-2

(Best.-Nr. 402)

nur **49,- DM***



bringt Ihrer Tastatur das Zaubern bei

Legen Sie ganze Texte auf 1 Taste!

Aktivieren Sie Befehlsfolgen mit 1 Tastendruck!

Recorderfunktion zur Makroaufzeichnung · mit "Copy Cursor" wird Bildschirmtext zum Makro · gezieltes Abspeichern mit Kommentaren · komfortabelste Editiermöglichkeiten aller Makros · verschiedene Makros auf 1 Taste · "Chaining": Makros rufen sich gegenseitig auf · "Sendkeys": Makros in der Batch-Datei · Hyperkey läuft im Hintergrund anderer Programme · mit Makros für Basic- und Pascalprogrammierer

mit TSR-Verwaltung

Auflisten aller geladenen speicherresidenten Programme · Angabe ihres Speicherplatzbedarfs und der verwendeten Interruptvektoren · Berechnung des noch zur Verfügung stehenden Speicherplatzes · Markieren von residenten Programmen beim Ladevorgang · Gefahrloses Entfernen
Alle Programme mit umfangreicher Dokumentation.

Best.-Nr. 229 5 1/4" Disk.

Best.-Nr. 235 3 1/2" Disk.

99,- DM*

(unverbindliche Preisempfehlung)

BCi DisAsm erschließt jedes Programm!

BCi DisAsm – Der intelligente Disassembler

BCi DisAsm ist ein intelligenter, selbsttätig dokumentierender MS-DOS Disassembler. Die ausgefeilten Routinen im BCi DisAsm analysieren in kürzester Zeit selbst komplexen und umfangreichen Programmcode und zeigen dabei potentielle Problemunkte auf.

BCi DisAsm – Das Lernmittel

Sie brauchen kein Spezialist in Assembler-Programmierung zu sein. BCi DisAsm produziert aus jedem beliebigen Programm ihres IBM-kompatiblen Computers ein gut dokumentiertes, leicht verständliches Assemblerlisting.

BCi DisAsm – Einfach in der Handhabung

BCi DisAsm kann sowohl über Parameter von der DOS-Kommandozeile aus als auch interaktiv über Menüs gesteuert werden. Es besteht keine Notwendigkeit, neue Befehle zu lernen oder ständig im Handbuch nachzuschlagen.

Minimale Systemanforderungen

256 kB RAM, 8088/8086/80186/80286 oder 80386 CPU, MS-DOS oder PC-DOS Version 2.0 oder größer

BCi DisAsm wird Ihnen bei den folgenden Schritten helfen:

1. Lernen Sie Assembler-Programmierung, wenn Sie möchten.
2. Entdecken Sie, warum Programm XYZ auf ihrem Rechner nicht läuft und tun Sie es was dagegen.
3. Ändern oder entfernen Sie Programmteile.
4. Sichern Sie sich gegen destruktive Programme ab. Finden Sie einen "Virus" und entfernen Sie ihn.
5. Integrieren Sie Code aus kompilierten Programmen in eigene Anwendungen.
6. Erhöhen Sie die Kompatibilität Ihrer Software.
7. Passen Sie Software an verschiedene MS-DOS-Versionen an.

© 1987 BCI, West Germany und USA · Lotus ist eingetragenes Warenzeichen der Lotus Corp.
MS-DOS ist eingetragenes Warenzeichen der Microsoft Corp.

Best.-Nr. 227 5 1/4" Disk.

Best.-Nr. 233 3 1/2" Disk.



Intelligenter Disassembler Version 1.0

199,- DM*

(unverbindliche Preisempfehlung)

PASCAL SONDERDRUCK

Das Beste aus PASCAL 1:1/86 bis 6/87

Aus dem Inhalt:

Kurvendiskussion · Mathematische Funktionen selbst erweitert · Rechnen mit komplexen Zahlen in Turbo Pascal · Gleichungssysteme und Ausgleichspolynome · Strings unter Wirth'schem Standard selbstgemacht · Press & Unpress, Dateien komprimiert abspeichern · GSX – Die unbekannte Größe bei CP/M 3.0 · Source-Code Referenz Lister · Dreidimensionale Darstellungen in Pascal ST+ (Atari ST) · Bibliothek zum Rechnen mit Matrizen in Turbo Pascal · Simulation in Pascal, ein Fahrkartenaufzählgerät · Ähnlichkeitsbestimmung von Zeichenketten · Pascal Colleg · Vom Punkt zur dritten Dimension · Zweidimensionale Meßwertdarstellung in Pascal · Plotter-Simulator für den Matrixdrucker · Externe Kommandos in Turbo Pascal 3.0 · Menügesteuerte Druckereinstellung · Datenfernübertragungs-Programm in Pascal · HGR-Copy: Bildschirmausschnitte und Zoom bei Hardcopies · Interrupts · Der Case-Trick, PEEK und POKE für Pascal-Programmierer

Alle Programme sind in der Praxis getestet, Nachträge und Verbesserungen wurden eingearbeitet. Sie sind, mit einer Ausnahme, unter Turbo Pascal 3.0 lauffähig und eignen sich für alle Hardware und Betriebssysteme vom Homecomputer bis zum Mini.

PASCAL Sonderdruck Best.-Nr. 345

25,- DM

Alle Programme sind auf Datenträger in der abgedruckten Version erhältlich:

PASCAL Sonderdruck Volume 1 und 2

MS-DOS 5 1/4" Best.-Nr. 346 **35,- DM** Atari ST 3 1/2" Best.-Nr. 348 **35,- DM**

MS-DOS 3 1/2" Best.-Nr. 347 **35,- DM** andere Formate auf Anfrage

ware, besitzt GW-Basic die Fähigkeit, mit dem Lichtgriffel effizient umzugehen. Zumeist muß das Programm ständig darüber informiert sein, wo die momentane Position des Lichtgriffels ist, zum Beispiel, um dort eine Markierung zu zeigen. Dafür löst der Rechner ein Pen-Ereignis aus, das mit

```
on pen gosub <Zeile>
abgefangen wird.
```

Nach der Pen-On-Aktivierung begibt sich der Rechner immer dann zur vereinbarten Zeile, wenn eine Lichtgriffel-Aktivität vorliegt. Sobald dieser an- oder abgesetzt wird, seine Position ändert oder der Knopf gedrückt ist, tritt das Ereignis ein. In der Ereignisroutine erfährt das Programm durch die Funktion »pen(n)« genau den Zustand des Lichtgriffels (Tabelle 8). Das Programm »onpen« (Listing

9) verwendet das On-Pen-Ereignis, um ständig die Spalten- und Zeilenposition des Lichtgriffels in der Text-Matrix anzuzeigen. Das Programm endet nach einem Druck auf den Lightpen-Knopf.

Auch wenn Sie keinen Lichtgriffel besitzen – zumindest wissen Sie jetzt in der Theorie, wie er in GW-Basic abgefragt wird.

(Dieter Bäumler/mk)

```
[V12] 10 KEY OFF:CLS
[YU4] 20 LINE INPUT "Dateiname: ";DAT$
[X23] 30 ON ERROR GOTO 1000
[U54] 40 OPEN DAT$ FOR INPUT AS #1
[ES2] 50 ON ERROR GOTO 0
[Z14] 60 IF FEHLER=0 THEN GOTO 80
[PL3] 70 FEHLER=0:BEEP:GOTO 20
[W2] 80 WHILE NOT EOF(1)
[EG3] 90 LINE INPUT #1,ZEILE$
[XH2] 100 PRINT ZEILE$
[E61] 110 WEND
[X92] 120 CLOSE 1:END
[BL4] 1000 FEHLER=ERR:RESUME NEXT
```

Listing 1. »type.bas« arbeitet beim Öffnen der Datei mit dem Error-Trapping

```
[ST2] 1 ON ERROR GOTO 10
[DZ0] 2 NEXT
[6B0] 3 (
[KC1] 4 RETURN
[631] 5 READ A
[DU1] 6 A$=CHR$(256)
[FA1] 7 A=99*99
[772] 8 DIM A(1000,1000)
[UH4] 10 ON ERR GOTO 12,13,14,15,16,17,19
[YC3] 11 ON ERROR GOTO 0:END
[VN3] 12 CLS:I=31:RESUME NEXT
[B53] 13 I=I+1:RESUME NEXT
[K43] 14 O=1-O:RESUME NEXT
[YG4] 15 COLOR 7-O*7,O*7:RESUME NEXT
[ML4] 16 PRINT CHR$(I);:RESUME NEXT
[XN3] 17 IF I<255 THEN RESUME 3
[L82] 18 RESUME NEXT
[FK6] 19 PRINT:PRINT "Das wars...":RESUME NEXT
```

Listing 2. »error.bas« – Die Programm-Steuerung wird durch bewußte Fehler vorgenommen

```
[904] 1 'Programmname: mint.bas
[534] 2 'Autor: Dieter Bäumler
[4P6] 3 '(c)1988 Redaktionsbüro Everts&Hagedorn
[P63] 4 'DOS International
[CE3] 10 ON TIMER(1) GOSUB 400
[WM3] 20 RANDOMIZE TIMER:CLS
[144] 30 R=INT(RND*10000):KEY OFF
[PP2] 40 RS=MID$(STR$(R),2)
[W54] 50 RS=STRING$(4-LEN(RS),48)+RS
[K74] 60 I=I+1:LOCATE I,1:COLOR 7,0
[R44] 70 PRINT STRING$(4-LEN(RS),48);
[VL4] 80 PRINT MID$(STR$(I),2);". ";S=1
[YP2] 90 TIMER ON:BS=""
[G43] 100 LOCATE I,S+4,1,1,11
[KT1] 110 A$=INKEY$
[G04] 120 IF ENDE=1 THEN GOTO 370
[9F3] 130 IF A$="" THEN GOTO 110
[AR4] 140 A=INSTR("0123456789",A$)*-(S<5)
[SC3] 150 IF A=0 THEN GOTO 180
[3K5] 160 PRINT A$;MID$(BS,S,1)=A$:S=S-(S<5)
[4P1] 170 GOTO 100
[1C4] 180 IF A$=CHR$(8) THEN GOTO 320
[PB5] 190 IF A$<>CHR$(13) THEN BEEP:GOTO 100
[F44] 200 IF S<5 THEN BEEP:GOTO 100
[SM2] 210 TIMER OFF:Z=0
[NE3] 220 T=0:G=0:FOR O=1 TO 4
[Y74] 230 X=(MID$(RS,O,1)=MID$(BS,O,1))
[MN4] 240 IF X=-1 THEN T=T-X:GOTO 260
[S85] 250 G=G+SGN(INSTR(RS,MID$(BS,O,1)))
[Y83] 260 NEXT O:PRINT " ";
[BD5] 270 COLOR 13,0:PRINT STRING$(T,254);
[WD3] 280 IF T=4 THEN GOTO 350
[765] 290 COLOR 7,0:PRINT STRING$(G,254)
[VE3] 300 IF I=20 THEN GOTO 370
[EY5] 310 LOCATE 23:PRINT SPACES(80);:GOTO 60
[D44] 320 IF S=1 THEN BEEP:GOTO 100
[765] 330 S=S-1:MID$(BS,S,1)=" ":LOCATE ,S+4
[6B3] 340 PRINT " ";:GOTO 100
[YY5] 350 PRINT:PRINT "Sie haben gewonnen"
[4D4] 360 COLOR 7,0:TIMER OFF:END
```

```
[1Z5] 370 PRINT:PRINT "Sie haben verloren"
[BD6] 380 PRINT "Die Lösung hätte gelaute: ";
[AP5] 390 PRINT RS:COLOR 7,0:TIMER OFF:END
[E44] 400 Z=Z+1:X=POS(1):Y=CSRLIN
[G45] 410 LOCATE 23,Z,0:PRINT CHR$(219);
[ZL5] 420 LOCATE Y,X,1:ENDE=-(Z=80):RETURN
```

Listing 3. »mint.bas« zeigt die vergangene Zeit für jeden Zug mit einem Balken an

```
[FW4] 10 KEY OFF:CLS:RANDOMIZE TIMER
[KN4] 20 DX(1)=0:DX(2)=-1:DX(3)=1:DX(4)=0
[NP4] 30 DY(1)=-1:DY(2)=0:DY(3)=0:DY(4)=1
[BS2] 40 WX=INT(RND*80)+1
[VR2] 50 WY=INT(RND*24)+1
[BS2] 60 SX=INT(RND*80)+1
[VR2] 70 SY=INT(RND*24)+1
[B93] 80 ON KEY(1) GOSUB 1010
[K93] 90 ON KEY(12) GOSUB 1020
[6C3] 100 ON KEY(13) GOSUB 1030
[CC3] 110 ON KEY(14) GOSUB 1040
[7T4] 120 LOCATE SY,SX:PRINT CHR$(2);
[674] 130 LOCATE WY,WX:PRINT "e";
[E63] 140 A=2:FOR I=11 TO 14
[122] 150 KEY (I) ON
[YG1] 160 NEXT I
[F33] 170 ZUG=INT(RND*4)+1
[N54] 180 LOCATE WY,WX:PRINT " ";
[KP2] 190 WX=WX+DX(ZUG)
[ZN2] 200 WY=WY+DY(ZUG)
[873] 210 IF WX=81 THEN WX=1
[U63] 220 IF WY=25 THEN WY=1
[393] 230 IF WX=0 THEN WX=80
[C93] 240 IF WY=0 THEN WY=24
[E74] 250 LOCATE WY,WX:PRINT "e";
[4R4] 260 IF SPZUG<>0 THEN GOSUB 1070
[0W4] 270 A=0:FOR I=1 TO 400:NEXT I:A=2
[PN5] 280 IF WX=SX AND WY=SY THEN GOTO 300
[8R1] 290 GOTO 170
[4P1] 300 A=1:BEEP
[F15] 310 LOCATE 23,1:PRINT "Erwischt !"
[2D5] 320 FOR I=11 TO 14:KEY(I) OFF:NEXT I
[331] 330 END
[4T5] 1000 REM ** Tastatur-Unterbrechung **
[293] 1010 SPZUG=1:GOTO 1050
[793] 1020 SPZUG=2:GOTO 1050
[C93] 1030 SPZUG=3:GOTO 1050
[CV1] 1040 SPZUG=4
[2P3] 1050 IF A=2 THEN RETURN
[865] 1060 IF A<>0 THEN SPZUG=0:RETURN
[AA4] 1070 LOCATE SY,SX:PRINT " ";
[K93] 1080 SX=SX+DX(SPZUG)
[LK5] 1090 IF SX>80 AND SX<81 THEN GOTO 1110
[X83] 1100 SX=SX-DX(SPZUG)
[293] 1110 SY=SY+DY(SPZUG)
[4K5] 1120 IF SY>0 AND SY<25 THEN GOTO 1140
[K93] 1130 SY=SY-DY(SPZUG)
[BV1] 1140 SPZUG=0
[D35] 1150 LOCATE SY,SX:PRINT CHR$(2);
[LV1] 1160 RETURN
```

Listing 4. »willi.bas« – Die Cursor-Tasten werden abgefangen und dienen zur Steuerung der Spielfigur

```
[UZ2] 10 ON KEY(1) GOSUB 100
[PU2] 20 KEY OFF:KEY(1) ON
[AM4] 30 LOCATE ,1,2,10:PRINT CHR$(12);
[ZN4] 40 OPEN "com1:1200,n,8,1,pe" AS #1
[PY2] 50 ON COM(1) GOSUB 110
[HK1] 60 COM(1) ON
[L05] 70 A$=INKEY$:IF A$="" THEN GOTO 70
[E22] 80 PRINT #1,A$;
[XD1] 90 GOTO 70
[654] 100 KEY(1) OFF:COM(1) OFF:END
[5R3] 110 BS=INPUT$(256-LOF(1),1)
[YT3] 115 L=LEN(BS):FOR I=1 TO L
[0F2] 117 Z$=MID$(BS,I,1)
[XG4] 120 IF Z$<>CHR$(10) THEN GOTO 130
[FA4] 122 X=POS(0):PRINT:LOCATE ,X
[355] 123 PRINT CHR$(28);CHR$(29);:GOTO 180
```



```
[AH4] 130 IF Z$<>CHR$(13) THEN GOTO 140
[UR5] 132 LOCATE ,2:PRINT CHR$(29);:GOTO 180
[PC4] 140 IF Z$=CHR$(8) THEN GOTO 160
[KA3] 150 PRINT Z$;:GOTO 180
[D24] 160 IF POS(0)=1 THEN GOTO 180
[994] 170 PRINT CHR$(29);" ";CHR$(29);
[6T2] 180 NEXT I:RETURN
```

Listing 5. »term.bas« – Das Terminal-Programm arbeitet mit der On-Com-Unterbrechung

```
[572] 10 PLAY ON:CLS
[M74] 20 PLAY "mlmb110o3":GOSUB 100
[K92] 30 FOR I=1 TO 9999
[UP1] 40 PRINT I;
[WAL] 50 NEXT I
[HF2] 60 PLAY OFF:END
[Z84] 100 PLAY"cgd#cdcdcd#cgd#cdcdcd#"
[7X4] 110 ON PLAY(15) GOSUB 120:RETURN
[UK4] 120 PLAY"cg#d#cdcdcd#cg#d#cdcdcd#"
[HX4] 130 ON PLAY(17) GOSUB 100:RETURN
```

Listing 6. »musik.bas« – Beliebige lange Musikstücke im Hintergrund abspielen

```
[SR4] 1 'Programmname: asteroid.bas
[534] 2 'Autor: Dieter Bäumler
[4P6] 3 '(c)1988 Redaktionsbüro Everts&Hagedorn
[P63] 4 'DOS International
[7C5] 100 CLS:RANDOMIZE TIMER:KEY OFF
[PB3] 110 TX=40:TY=12:ANZ=10
[785] 120 DIM X(20),Y(20),KX(20),KY(20),M(20)
[SF4] 130 DIM Z(20),SX(20),SY(20),X(20)
[VV4] 140 ON STRIG(0) GOSUB 3000 'Feuer
[T75] 150 XX(0)=0:YY(0)=1:XX(1)=1:YY(1)=0
[G45] 160 XX(2)=0:YY(2)=1:XX(3)=1:YY(3)=0
[TF4] 170 F(0)=24:F(1)=26:F(2)=25:F(3)=27
[B51] 180 RS=0
[EE5] 190 LOCATE TY,TX:PRINT CHR$(F(RS));
[VA2] 200 STRIG(0) ON
[PU4] 210 'jeder Figur ihre Position
[BH5] 220 FOR U=1 TO ANZ:GOSUB 1000:NEXT U
[ZY5] 230 FOR U=1 TO ANZ:LOCATE 1,1:PRINT PK
[HP5] 240 IF SCH>0 THEN GOSUB 8000 'Schuss ?
[7A4] 250 GOSUB 2000 'ziehe Figur u
[SN2] 260 JOYX=STICK(0)
[XN2] 270 JOYX=STICK(1)
[456] 280 IF ABS(JOYX)=ABS(JOYX) THEN GOTO 310
[9T4] 290 ON JOYX+2 GOSUB 6000,8100,5000
[SS4] 300 ON JOYX+2 GOSUB 4000,8100,7000
[U45] 310 IF MV=1 THEN GOSUB 9000 'Schub ?
[FP5] 320 IF DR=1 THEN GOSUB 10000 'Drehung ?
[Z03] 330 NEXT U:GOTO 230
[GH0] 340 '
[WZ5] 1000 X=INT(RND*80)+1 'x- und y für figur
[AK5] 1010 Y=INT(RND*24)+1 'u ermitteln und
[0S5] 1020 E1=INT(RND*2) 'ob x oder y 0 oder
[8H5] 1030 E2=INT(RND*2) '80 (24) sein soll.
[HH3] 1040 IF E1 THEN GOTO 1070
[674] 1050 IF E2 THEN X=80 ELSE X=1
[HW1] 1060 GOTO 1080
[074] 1070 IF E2 THEN Y=24 ELSE Y=1
[FE5] 1080 IF SCREEN(Y,X)=64 THEN GOTO 1000
[1A4] 1090 'steht schon einer da ?
[TD5] 1100 LOCATE Y,X:PRINT "e"; 'mach hin
[RP0] 1110 :
[PF5] 1120 DX=TX-X:SY=SGN(DX):DX=ABS(DX)
[EG5] 1130 DY=TY-Y:SY=SGN(DY):DY=ABS(DY)
[0X4] 1140 'x,y-Abstand und -Richtung
[D92] 1150 K=- (DY>DX)
[Z65] 1160 IF DX=0 THEN M=1 ELSE M=DY/DX
[ZP6] 1170 'm=Steigungsfaktor der Bahngeraden
[GR3] 1180 IF K=0 THEN GOTO 1200
[D75] 1190 IF DY=0 THEN M=1 ELSE M=DX/DY
[K56] 1200 SX(U)=SX:SY(U)=SY:KY(U)=Y:KX(U)=X
[Z25] 1210 Y(U)=Y:X(U)=X:M(U)=M:Z(U)=0:K(U)=K
[6V1] 1220 RETURN
[2R0] 1230 :
[ZM5] 2000 Z(U)=Z(U)+1 'z=Schrittzahl(u)
[554] 2010 IF K(U)=1 THEN GOTO 2070
[FH6] 2020 'Berechnung des nächsten Schritts
[DV3] 2030 'auf der Bahngeraden:
[8K3] 2040 KX=X(U)+SX(U)*Z(U)
[VW5] 2050 KY=Y(U)+SY(U)*INT(M(U)*Z(U)+.444))
[TW1] 2060 GOTO 2090
[YK3] 2070 KY=Y(U)+SY(U)*Z(U)
[XW5] 2080 KX=X(U)+SX(U)*INT(M(U)*Z(U)+.444))
[715] 2090 LOCATE KY(U),KX(U):PRINT " ";
[WP3] 2100 'Am Rand angelangt ?
[4C5] 2110 IF KX>80 OR KY<1 THEN GOTO 1000
[HC5] 2120 IF KY>24 OR KY<1 THEN GOTO 1000
[S65] 2130 KP=SCREEN(KY,KX) 'Wer ist da ?
[BC4] 2140 LOCATE KY,KX:PRINT "e";
[105] 2150 IF KP<30 THEN SOUND 50,10:END
[LY3] 2160 'Stand da wer ? -->Ende
[8P4] 2170 KY(U)=KY:KX(U)=KX:RETURN
[NR0] 2180 :
[CV3] 3000 IF SCH>0 THEN RETURN
[NC6] 3010 'Ist Schuss unterwegs ? -->Zurueck
```

```
[YA5] 3020 SCH=1:SS=RS:SCHX=TX:SCHY=TY
[CD5] 3030 'Position und Schussrichtung
[CV1] 3040 RETURN
[8R0] 3050 :
[NS5] 4000 TMPX=TX+XX(RS):TMPY=TY+YY(RS)
[ZS5] 4010 IF TMPX<1 OR TMPX>80 THEN RETURN
[9T5] 4020 IF TMPY<1 OR TMPY>24 THEN RETURN
[LZ5] 4030 'sind die Werte über den Schirm ?
[WC6] 4040 MV=1:RETURN 'Sonst wurde gefahren
[AR0] 4050 :
[2U5] 5000 TMPRS=RS-1 'Drehung nach rechts
[3L4] 5010 IF TMPRS=-1 THEN TMPRS=3
[XK2] 5020 DR=1:RETURN
[4R0] 5030 :
[KR5] 6000 TMPRS=RS+1 'Drehung nach links
[MG4] 6010 IF TMPRS=4 THEN TMPRS=0
[XK2] 6020 DR=1:RETURN
[6R0] 6030 :
[6Y5] 7000 TMPX=INT(RND*80)+1 'Neue Position
[YG5] 7010 TMPY=INT(RND*24)+1 'per Zufall
[D56] 7020 WD=SCREEN(TMPX,TMPY) 'Ist da wer ?
[FK5] 7030 IF WD<>32 THEN GOTO 7000 'nochmal
[2D6] 7040 MV=1:RETURN 'Sonst wurde gefahren
[GR0] 7050 :
[P44] 8000 IF SCH=1 THEN GOTO 8030
[EG4] 8010 'Ist der Schuss frisch ?
[X05] 8020 LOCATE SCHY,SCHX:PRINT " ";
[W93] 8030 SCHY=SCHY+YY(SS)
[S93] 8040 SCHX=SCHX+XX(SS)
[E55] 8050 IF SCHX<1 OR SCHX>80 THEN 8210
[S55] 8060 IF SCHY<1 OR SCHY>24 THEN 8210
[814] 8070 WD=SCREEN(SCHY,SCHX)
[7K5] 8080 IF WD=64 THEN GOTO 8120 'Treffter
[EV5] 8090 SCH=2:LOCATE SCHY,SCHX:PRINT " ";
[9V1] 8100 RETURN
[5R0] 8110 :
[AY5] 8120 LOCATE SCHY,SCHX:PRINT " "; 'Leer
[W85] 8130 TR=0:FOR I=1 TO ANZ
[SZ5] 8140 IF SCHX<>KX(I) THEN GOTO 8170 'ist
[5V5] 8150 IF SCHY<>KY(I) THEN GOTO 8170 'es ?
[4G6] 8160 TR=I:I=ANZ 'tr ist der Getroffene
[DA1] 8170 NEXT
[NG6] 8180 'Neue Position fuer den Getroffenen
[9K5] 8190 TMPU=U:U=TR:GOSUB 1000:U=TMPU
[X45] 8200 PK=PK+1 'Punkte hochzaehlen
[LM2] 8210 SCH=0:RETURN
[CR0] 8220 :
[2X5] 9000 LOCATE TY,TX:PRINT " "; 'Bewegung
[FH3] 9010 TY=TMPY:TX=TMPX
[DR5] 9020 WD=SCREEN(TY,TX) 'Steht da wer ?
[4P5] 9030 LOCATE TY,TX:PRINT CHR$(F(RS));
[B05] 9040 IF WD=64 THEN SOUND 50,10:END
[UM2] 9050 MV=0:RETURN
[SR0] 9060 :
[5S3] 10000 RS=TMPRS 'Drehung
[US5] 10010 LOCATE TY,TX:PRINT CHR$(F(RS));
[XG2] 10020 DR=0:RETURN
```

Listing 7. »asteroid.bas« – Das Spiel verwendet die On-Strig-Anweisung, um den Feuerknopf des Joystick abzufangen

```
[1D5] 140 'Cursor-Tasten zur Steuerung
[DY4] 141 on key(1) gosub 3000 'F1 Feuer
[ZE4] 142 on key(11) gosub 4000 'Schub
[7Y5] 143 on key(12) gosub 5000 'Rechtsdrehung
[VV5] 144 on key(13) gosub 6000 'Links drehung
[X55] 145 on key(14) gosub 7000 'Teleport
[FT1] 146 key(1) on
[KC5] 147 for i=11 to 14:key(i) on:next i
[DP1] 260 goto 310
```

Listing 8. Wenn Sie »ovr.bas« nach dem Laden von »asteroid« mit dem Merge-Befehl lesen, verwendet das Spiel zur Steuerung die Cursor-Tasten

```
[V12] 10 KEY OFF:CLS
[AT2] 20 ON PEN GOSUB 100
[551] 30 PEN ON
[ZN1] 40 LOCATE 1,1
[6T1] 50 PRINT X,Y
[ZH3] 60 IF ENDE=0 THEN GOTO 40
[FB1] 70 PEN OFF
[BP1] 80 LOCATE 2,1
[FT0] 90 END
[B72] 100 ENDE=PEN(0)
[SK3] 110 LOCATE Y,X:PRINT " ";
[VP1] 120 Y=PEN(6)
[OR1] 130 X=PEN(7)
[5F4] 140 IF X=0 OR Y=0 THEN GOTO 160
[GM3] 150 LOCATE Y,X:PRINT "+";
[9Y1] 160 LOCATE 1,1
[DM1] 170 RETURN
```

Listing 9. »onpen.bas« zeigt die Position des Lichtgriffels in Koordinaten an, wobei der On-Pen-Befehl jede Lichtgriffel-Aktion erfasst

Basic von A bis Z

Die Programmiersprache Basic erfreut sich nicht nur im Heimcomputerbereich, sondern inzwischen auch bei den PC-Besitzern großer Beliebtheit. Das folgende kleine Basic-Lexikon verschafft Ihnen einen Überblick über die Bedeutung einiger grundlegender Begriffe und Fachwörter rund um diese Programmiersprache.

Abbruch: Der → *Interpreter* → *GW-Basic* erlaubt Ihnen den Abbruch des Programmablaufs durch das Drücken der Tastenkombination [Ctrl-Break]. Um nach erfolgreicher Fehlerbeseitigung die Ausführungsgeschwindigkeit eines Programms zu erhöhen und Fehlbedienungen auszuschließen, stellt man diese Unterbrechungsmöglichkeit gewöhnlich ab.

Alphanumerische Variable: Eine alphanumerische → *Variable* ist ein Platzhalter, dessen Inhalt aus beliebigen Zeichen bestehen kann. Im Gegensatz zu → *numerischen Variablen* sind für alphanumerische Variablen also auch Buchstaben und Sonderzeichen erlaubt. Den Inhalt einer alphanumerischen Variablen bezeichnet man als → *Zeichenkette* oder → *String*.

Anschluß: Ein- und Ausgabegabekanal zu und von einem Computer für die Ansteuerung von peripheren Geräten. (→ *Port*)

ANSI: ANSI (»American National Standard Institute«) ist ein amerikanisches Institut, das ähnlich dem deutschen DIN-Institut Industriennormen festlegt. Das ANSI hat beispielsweise den → *ASCII-Zeichensatz* und den Standardbefehlssatz von → *Basic* definiert.

Anweisung: Folge von → *Befehlen*, die den Computer veranlassen, bestimmte Operationen durchzuführen.

Array: → *Feld*

ASCII: ASCII (»American Standard Code for Information Interchange«) ist ein 7-Bit-Zeichencode, der von den meisten Computern und Programmiersprachen für den Austausch von Informationen benutzt wird. ASCII wurde vom → *ANSI* als Standardzeichensatz festgelegt.

Basic: Basic (»Beginners All Purpose Symbolic Instruction Code«) ist die weitverbreitetste Programmiersprache der Welt. Sie wurde in den sechziger Jahren am Dartmouth-College von John Kemeny und Thomas Kurtz zur Demonstration von grundlegenden Programmstrukturen konzipiert. Basic gilt als zeitsparendes Mittel zur Programmentwicklung und ist eine

der am leichtesten erlernbaren Programmiersprachen. Im PC-Bereich hielt Basic durch den Microsoft-Standard → *GW-Basic* Einzug. Neben der großen Verbreitung im Heimbereich hat Basic auch im semiprofessionellen und professionellen Bereich durch die Entwicklung von modernen Basic-Dialekten wie → *Quick Basic* und → *Turbo Basic* einen bemerkenswerten Aufschwung erlebt.

Bedingte Ausführung: Der Befehl »if .. then .. else ..« veranlaßt Basic, einen Programmteil nur dann auszuführen, wenn eine bestimmte Bedingung erfüllt ist. Die Anweisung

```
if a = 0 then b = 1 else b = 2
```

weist beispielsweise der Variablen »b« nur dann den Wert 1 zu, wenn die Variable »a« auf Null gesetzt ist. Anderenfalls (»else«) erhält »b« den Wert 2. Für die Überprüfung einer Bedingung können sowohl die logischen → *Vergleichsoperationen* als auch die Operatoren »=«, »<«, »>«, »<=«, »>=« und »<>« benutzt werden.

Die If-Then-Else-Schleife erlaubt die bedingte Ausführung von Programmteilen

Befehl: Ein Basic-Befehl oder → *Kommando* ist ein vordefiniertes Wort aus einer Folge von Zeichen, das den PC veranlaßt, bestimmte Operationen wie beispielsweise das Löschen des Bildschirms durch »cls« durchzuführen. Jeder Befehl wird von einem Interpreter in ein → *Token* übersetzt.

Begrenzungszeichen: Begrenzungszeichen werden benutzt, um den Anfang beziehungsweise das Ende einer Dateneinheit auf einem Speichermedium zu markieren. Üblicherweise verwendet man Kommata, Leerzeichen, Punkte oder Semikolons als Begrenzungszeichen zur Trennung und Gliederung von Daten. In → *Basic* hat sich der Einsatz des Kommas und des Enter-Zeichens (→ *ASCII-Wert* 13) als Begrenzungszeichen durchgesetzt.

Boolsche Logik: Unter diesem Begriff faßt man alle logischen → *Vergleichsoperationen* der Programmiersprache → *Basic* zusammen.

Compiler: Ein Compiler wie → *Quick Basic* oder → *Turbo Basic* übersetzt einen Quelltext nach erfolgreicher Fehlerbeseitigung in einen maschinenspracheähnlichen Code (»Objekt-Code«) oder sogar direkt in Maschinensprache. Ein auf diese Weise übersetztes Programm erreicht wesentlich höhere Ablaufgeschwindigkeiten als ein → *Interpreter* wie → *GW-Basic*.

Direktmodus: *GW-Basic* bietet die Annehmlichkeit, Befehle direkt auf den Bildschirm schreiben und so gleich ausführen zu können. Der eingetippte Befehl wird nach Betätigen der Eingabetaste sofort aufgeführt. Im Gegensatz zum → *Programmmodus* lassen sich im Direktmodus allerdings längere Anweisungen, die die Länge einer Programmzeile überschreiten, nicht ausführen.

Feld: Ein Feld oder → *Array* ist eine strukturierte Sammlung von Daten. Ein Feld setzt sich aus mehreren → *Variablen* zusammen. Die Vorteile eines Variablenfeldes sind auffällig: Einerseits kann jedes Element des Feldes unter demselben Variablennamen angesprochen werden; der einzige Unterschied besteht im Indexwert. Andererseits sind Sie in der Lage, jedes Feldelement genauso zu bearbeiten wie eine Einzelvariable vom gleichen Variablentyp.

Listing 1 zeigt ein praktisches Beispiel zum Umgang mit Feldern. In Zeile 10 wird dem Computer über den Dim-Befehl mitgeteilt, daß er unter dem Variablennamen »zahl« eine Gruppe von maximal 10 Werten zusammenfassen soll. Der → *Interpreter* reserviert darauf unter dem Namen »zahl« Speicherplatz für maximal 10 Werte. Die einzelnen Werte sind durchnummeriert, wobei die Nummern in den Klammern nach dem Variablennamen angegeben werden. Die Nummer für die einzelnen Feldelemente wird auch als → *Index* bezeichnet.

Fließkomma-Variablen: Die Fließkomma-Variablen sind eine Teilmenge der → *numerischen Variablen*. Fließkomma-Variablen können im Gegensatz zu den → *Integer-Variablen* nicht nur mit ganzen, sondern auch mit rationalen Werten arbeiten. Unter → *GW-Basic* werden Fließkomma-Variablen entweder Werte mit einfacher (4 Byte) oder doppelter Genauigkeit (8 Byte) zugewiesen. (→ *Variable*)

Grafikkarten: Im Laufe der Zeit sind viele PC-Grafikkarten auf den Markt gekommen (Monochromadapter (MDA), Hercules, CGA, EGA und VGA). In bezug auf die Grafikanpassung gibt es bei den verschiedenen

Basic-Versionen nur bei der Hercules-Grafikkarte Probleme, weil diese nicht von allen Basic-Dialekten berücksichtigt wird. Die Farbgrafikkarten CGA, EGA und VGA werden dagegen von den neuesten Versionen der verschiedenen Basic-Dialekte ausnahmslos unterstützt.

GW-Basic: GW-Basic ist der bekannteste → Interpreter für PCs. Er ist meistens in Kombination mit dem Betriebssystem MS-DOS im Lieferumfang eines PC enthalten und eignet sich besonders gut für den schnellen Einstieg in das Programmieren. Der Wortbestandteil »GW« (»Graphics and Windows«) weist darauf hin, daß GW-Basic schon in seiner ursprünglichen Version Grafikbefehle enthält und (eingeschränktes) Arbeiten mit Fenstern erlaubt.

GW-Basic gilt als Basic-Standard im PC-Bereich. Auf dem Befehlssatz dieses Basic-Interpreters basieren die modernen Compiler wie beispielsweise → Quick Basic und → Turbo Basic.

Index: Der Index bestimmt innerhalb eines → Feldes ein Feldelement. Dieses Element wird wie eine → Variable angesprochen.

Integer-Variablen: Integer-Variablen sind eine Teilmenge der → numerischen Variablen. Integer-Variablen können nur mit ganzen Zahlen arbeiten. Unter → GW-Basic ist ihnen der Wertebereich von -32768 bis 32767 zugewiesen. (→ Variable)

Interpreter: Ein Interpreter wie → GW-Basic übersetzt ein Hochsprachenprogramm während der Ausführung Befehl für Befehl in Maschinensprache. Weil jedoch beim Ausführen eines Befehls neben der Übersetzung und den gesamten Fehlerabfang-Routinen auch noch die vom jeweiligen Befehl benötigten Systemroutinen aufgerufen werden müssen, ist die erreichbare Verarbeitungsgeschwindigkeit in den meisten Fällen vergleichsweise gering. Deshalb ist die Ablaufgeschwindigkeit eines Interpreters wesentlich langsamer als die eines → Compilers.

Interrupt-Programmierung: Mit Basic sind Sie in der Lage, die Abfrage gewisser Zustände wie beispielsweise der Tastatur in einen Interrupt einzubinden. Der Interrupt wird mehrere Male pro Sekunde aufgerufen. Dabei nimmt Basic jedesmal die vorher definierte Abfrage vor. Abhängig vom Ergebnis führt der Basic-Interpreter oder Compiler dann ein → Unterprogramm aus und setzt anschließend die Programmausführung im Hauptprogramm fort.

Kommando: → Befehl. Der Standardbefehlssatz von Basic wurde durch das → ANSI festgelegt und von Microsoft in → GW-Basic übernommen. Eine Aufstellung aller Basic-Befehle finden Sie in Tabelle 1. (→ Anweisung)

Listing: → Programm

Numerische Variablen belegen im Gegensatz zu den alphanumerischen Variablen eine feste Länge im Arbeitsspeicher

Numerische Variablen: Eine numerische → Variable ist ein Platzhalter, dessen Inhalt ausschließlich aus Zahlen, einem Vorzeichen und einem Dezimalpunkt bestehen darf. Im Gegensatz zu → alphanumerischen Variablen sind für numerische Variablen weder Buchstaben noch Sonderzeichen erlaubt.

Overlay: Wenn ein Programm so groß ist, daß es nicht mehr komplett in den Arbeitsspeicher des Computers paßt, verwendet man die Overlay-Technik. Overlay-Technik bedeutet, daß das Hauptprogramm in Einzelprogramme (Overlays) aufgeteilt wird. Das aktuelle → Programm lädt dann jeweils das nächste Einzelprogramm, das automatisch startet. Wenn gewünscht, können auch Variablen vom endenden an das geladene Einzelprogramm übergeben werden.

Port: Ein- und Ausgabekanal eines Computers zum Anschluß einer

Kommunikationsleitung. (→ Anschluß)

Programm: Die Zusammenfassung von mehreren Anweisungen, die dazu dienen, komplexe Probleme zu lösen. Oft wird ein Programm auch als → Listing bezeichnet, weil es im ausgedruckten Zustand aus einer Liste von Anweisungen besteht.

Programmodus: Im Gegensatz zum → Direktmodus werden die mit einer Zeilennummer beginnenden Basic-

Der Befehlssatz von GW-Basic 3.22

abs	hex\$	play
and	if	play
asc	imp	on/off/stop
atn	inkey\$	pmap
auto	inp	point
beep	input	poke
bload	input\$	pos
bsave	instr	preset
call	int	print
cdbl	ioctl	pset
chain	key	put
chdir	kill	randomize
chr\$	lcopy	rem
cint	left\$	renum
circle	len	reset
clear	line	restore
close	line input	right\$
cls	list	rmdir
color	llist	rnd
com	load	rset
common	loc	run
cont	locate	save
cos	lock	screen
crsln	lof	sgn
csng	log	shell
cvd	lpos	sin
cvi	lprint	sound
cvs	lset	space\$
def fn	merge	spc
def seg	mid\$	sqr
def usr	mkd\$	step
delete	mkdir	stick
dim	mkis\$	stop
draw	mks\$	str\$
edit	mod	strig
else	name	strig
end	next	on/off/stop
environ	new	string\$
environ\$	not	swap
eof	oct\$	system
eqv	on com gosub	tab
erase	on error goto	tan
erdev	on gosub	timer
erdev\$	on goto	on/off/stop
erl	on key gosub	then
err	on pen gosub	troff
error	on play gosub	tron
exp	on strig gosub	unlock
exterr	on timer gosub	using
field	open	usr
files	option base	val
fix	or	varptr
fn	out	varptr\$
for	paint	view
fre	palette	view print
get	pcopy	wait
gosub	peek	while
goto	pen	width
	pen on/off/stop	width lprint
		window
		write
		xor

Tabelle 1. Alle GW-Basic-Befehle auf einen Blick

Die Variablentypen von GW-Basic

Typ	Kennzeichen	Länge	Inhalt
Integer	%	2 Byte	Zahlen im Wertebereich von -32768 bis +32767
Single Precision	!	4 Byte	Fließkommazahlen mit einfacher Genauigkeit
Double Precision	#	8 Byte	Fließkommazahlen mit doppelter Genauigkeit
Variable Length String	\$	mindestens 4 Byte	Zeichenketten mit variabler Länge

Tabelle 2. GW-Basic arbeitet mit vier verschiedenen Variablentypen

Anweisungen (Programmzeilen) nicht direkt ausgeführt. Diese Zusammenfassung mehrerer Befehle nennt man → **Programm**.

Die Zeilennummer einer Programmzeile gibt die Position einer Anweisung innerhalb des Programms an. Im Gegensatz zum Direktmodus sind Sie im Programmodus in der Lage, die Folge von Basic-Anweisungen auf einem Datenträger dauerhaft zu speichern.

Prozedur: → **Unterprogramm**.

Quick Basic: Der moderne Basic-Dialekt Quick Basic von Microsoft ist der härteste Konkurrent von → **Turbo Basic**. Der → **Compiler** Quick Basic arbeitet mit einer optisch aufwendigen, menügeführten Benutzeroberfläche und kann mit GW-Basic konzipierte und im ASCII-Format abgelegte Programme bearbeiten.

Der integrierte Editor von Quick Basic ist wie der Editor von Turbo Basic Wordstar-kompatibel und überzeugt durch Schnelligkeit und seine vielfältigen Funktionen zur Bearbeitung des eingegebenen Quelltextes. Quick Basic präsentiert zu jedem benötigten Themenkomplex ein Pulldown-Menü und überzeugt durch ausgefeilte Fenstertechnik und Mausbedienung.

Rekursion: Eine Rekursion läßt sich in Basic durch ein → **Unterprogramm** realisieren, das sich immer wieder selbst aufruft. Eine Rekursion bricht ab, wenn entweder eine Abbruchbedingung erfüllt ist oder der interne Rechenstapel für weitere Wiederholungen nicht mehr ausreicht.

Das Beispiel in Listing 2 zeigt Ihnen den Aufbau einer Rekursion in GW-Basic. Das Programm berechnet die Fakultät einer Zahl (die Fakultät von 5 ist beispielsweise gleich $1 \times 2 \times 3 \times 4 \times 5 = 120$). Aufgrund der zugrundeliegenden Programmstruktur unterstützt → **GW-Basic** Rekursionen allerdings nicht zufriedenstellend. In den modernen Basic-Varianten wie → **Quick Basic** oder → **Turbo Basic** gehört diese Unterprogrammvariante aber schon zum Standard.

Relative Datei: In einer relativen Datei sind alle Datensätze mit einer festen Länge und gleicher Struktur auf dem Speichermedium abgelegt. Der Zugriff erfolgt über eine Satznummer. Da alle Datensätze den gleichen Aufbau vorweisen, ist der Zugriff wahlfrei.

Schleife: Eine Schleife ist gleichbedeutend mit einer Folge von Befehlen, die wiederholt werden, bis eine vorgegebene Bedingung erfüllt ist. Ein Beispiel für eine Schleife finden Sie in Listing 1. Eine Schleife wird in GW-Basic durch die Befehle »for« und »next« gebildet. Die Anweisun-

gen, die zwischen diesen beiden Befehlen stehen, werden ausgeführt, bis der Endwert der Schleife erreicht ist. Ist die Endbedingung erfüllt, dann fährt der → **Interpreter** mit der Programmausführung nach dem Next-Befehl fort. Um die Effizienz zu erhöhen, lassen sich mehrere Schleifen ineinander verschachteln.

Programmschleifen lassen sich auch mehrfach ineinander verschachteln

Sequentielle Datei: In einer sequentiellen Datei sind alle Datensätze mit einer variablen Länge und unterschiedlicher Struktur hintereinander auf dem Speichermedium abgelegt. Datensätze werden durch die ASCII-Zeichen 13 und 10 voneinander getrennt (→ **Begrenzungszeichen**).

Aufgrund der maximalen Länge eines Satzes von 255 Zeichen und der unflexiblen Struktur ist das Einsatzgebiet von sequentiellen Dateien unter → **GW-Basic** beschränkt.

Sprungbefehle: Sprungbefehle werden in Basic benutzt, um von der aktuellen Programmposition in einen anderen Programmteil zu verzweigen.

String: Das Wort »String« ist die englische Bezeichnung für → **Zeichenkette**.

Token: Der → **Interpreter** → **GW-Basic** übersetzt jeden Befehl in ein Token. So legt GW-Basic beispielsweise anstelle des Befehls »cls« einen bestimmten Byte-Wert im Arbeitsspeicher ab. Dieses Verfahren spart Platz sowohl im Arbeitsspeicher als auch auf dem externen Speichermedium.

Turbo Basic: Turbo Basic von Borland ist neben → **Quick Basic** der bekannteste → **Compiler** für die Programmiersprache → **Basic**. Turbo Basic unterstützt einen Großteil der Quick-Basic-Funktionen wie beispielsweise → **Rekursionen** oder verschiedene → **Grafikkarten**. Ein interessantes Leistungsmerkmal von Turbo Basic sind die zahlreichen Compiler-Befehle. Der Inline-Befehl erlaubt beispielsweise das direkte Einfügen von Maschinensprache in den Programmcode. Bemerkenswert ist die Fähigkeit von Turbo Basic, den Quelltext eines Basic-Programms bedingt zu übersetzen. Turbo Basic ist wie Quick Basic abwärts-kompatibel zu → **GW-Basic**.

Unterprogramm: Es gibt viele Situationen, in denen eine bestimmte Anweisungsfolge in einem → **Programm** mehrfach benutzt werden muß. In diesem Fall helfen zwar → **Schleifen** nicht weiter, doch läßt sich die An-

weisungsfolge als Unterprogramm realisieren. Ein Unterprogramm wird durch einen Gosub-Befehl aufgerufen und kehrt mit dem Return-Befehl an die Position zurück, die dem Unterprogrammaufruf folgt.

Unterprogramme verbessern die Struktur eines Listing und machen ein Programm übersichtlicher. Listing 2 zeigt Ihnen den Einsatz eines Unterprogramms in einer → **Rekursion**.

Variable: Variablen sind einheitliche Zeichengruppen, die als Platzhalter für bestimmte Werte in Programmen fungieren. Die Variablenart unterscheidet man mit Hilfe eines Kennzeichens voneinander, das den → **Variablentyp** angibt. Unter → **GW-Basic** ist die Namensgebung für Variablen Regeln unterworfen. Erstens dürfen Variablennamen höchstens 40 Zeichen lang sein und nur aus Buchstaben, Ziffern und Punkten bestehen. Zweitens dürfen Variablen nicht mit einer Zahl beginnen und drittens nicht den Namen eines Befehls führen.

→ **Alphanumerische Variablen** müssen zusätzlich die Bezeichnung »\$« am Ende des Variablennamens führen. Listing 2 demonstriert Ihnen, wie einer Variablen ein Wert zugewiesen wird.

Variablentyp: Der Typ einer → **Variablen** bestimmt, welche Werte ihr zugewiesen werden dürfen. Eine Zusammenfassung der unter → **GW-Basic** zur Verfügung stehenden Variablentypen und der zugehörigen Kennzeichen finden Sie in Tabelle 2. Die Größe von → **numerischen Variablen** und → **alphanumerischen Variablen** ist vom → **Interpreter** begrenzt. (→ **Integer-Variablen**, → **Fließkomma-Variablen**)

Vergleichsoperation: Eine Vergleichsfunktion wird benutzt, um zwei Datenfelder miteinander zu vergleichen und bei Bedarf das Ergebnis dieses Vergleichs in einem Datenfeld abzulegen. Unter → **GW-Basic** gibt es mehrere Vergleichsoperationen wie »and«, »eqv«, »in«, »not«, »or« und »xor«. Die Funktion dieser Vergleichsoperationen wurde 1864 von dem englischen Mathematiker George Boole festgelegt. (→ **Boolsche Logik**)

Verzeichnis: Ein Verzeichnis enthält Angaben über die auf einem Speichermedium in einem Inhaltsverzeichnis gespeicherten Dateien. Jeder Eintrag gibt den Namen, die Größe, das Erstellungsdatum und die Erstellungszeit einer Datei an.

Verzögerungsschleife: Eine Verzögerungsschleife ist eine → **Schleife**, die

die Programmausführung verlangsamt. Die Anweisung

```
for i = 1 to 1000 : next i
```

wird beispielsweise so lange durchlaufen, bis die Variable »q« den Wert 1000 erreicht und fährt erst dann mit der Programmausführung fort. Achten Sie darauf, daß die Wartezeit für diese Art von Schleifen computerab-

hängig ist. Auf einem AT ist die Ausführung schneller beendet als auf einem PC.

Verzweigung: Eine Verzweigung setzt die Programmausführung an der im Befehl angegebenen Position fort. Zu diesem Zweck benutzt man → Sprungbefehle, die die Art des Programmaufrufs bestimmen. Listing 3

zeigt Ihnen einige Beispiele für Sprungbefehle. (→ Bedingte Ausführung)

Zeichenkette: Eine Zeichenkette ist unter → Basic eine Folge von beliebigen ASCII-Zeichen.
(→ alphanumerische Variablen)

(Markus Zietlow/ma)

```
[H22] 10 DIM ZAHL(10)
[GF3] 20 FOR SCHLEIFE = 1 TO 10
[U68] 30 PRINT "Geben Sie bitte Zahl " ; SCHLEIFE ; "
      ein: ";
[WU3] 40 INPUT ZAHL(SCHLEIFE)
[7L2] 50 NEXT SCHLEIFE
[SF3] 60 FOR SCHLEIFE = 1 TO 10
[HH7] 70 PRINT "Zahl " ; SCHLEIFE ; ": " ; ZAHL(SCHLEIFE)
[DL2] 80 NEXT SCHLEIFE
```

Listing 1. Mit Basic lassen sich auch Variablenfelder verwalten

```
[941] 1070 END
[RN0] 1080 '
[FH2] 1090 'Rekursion
[UM0] 1100 '
[E45] 1110 FAKULTAET = FAKULTAET * ZAHL
[ER2] 1120 ZAHL = ZAHL - 1
[FM4] 1130 IF ZAHL > 0 THEN GOSUB 1110
[BV1] 1140 RETURN
```

Listing 2. Rekursionen sind in GW-Basic etwas unübersichtlich

```
[813] 1000 MENUE = AUSWAHL
[GL4] 1010 IF MENUE = 1 THEN GOTO 400
[EB6] 1020 IF MENUE = 2 THEN GOSUB 1200 : GOTO 400
[VY4] 1030 ON MENUE - 2 GOSUB 100, 200, 300
[DR1] 1040 GOTO 200
```

Listing 3. Für Verzweigungen existieren unter GW-Basic verschiedene Befehle

DOS INTERNATIONAL

Zur Ergänzung unseres jungen, engagierten Redaktionsteams suchen wir einen versierten

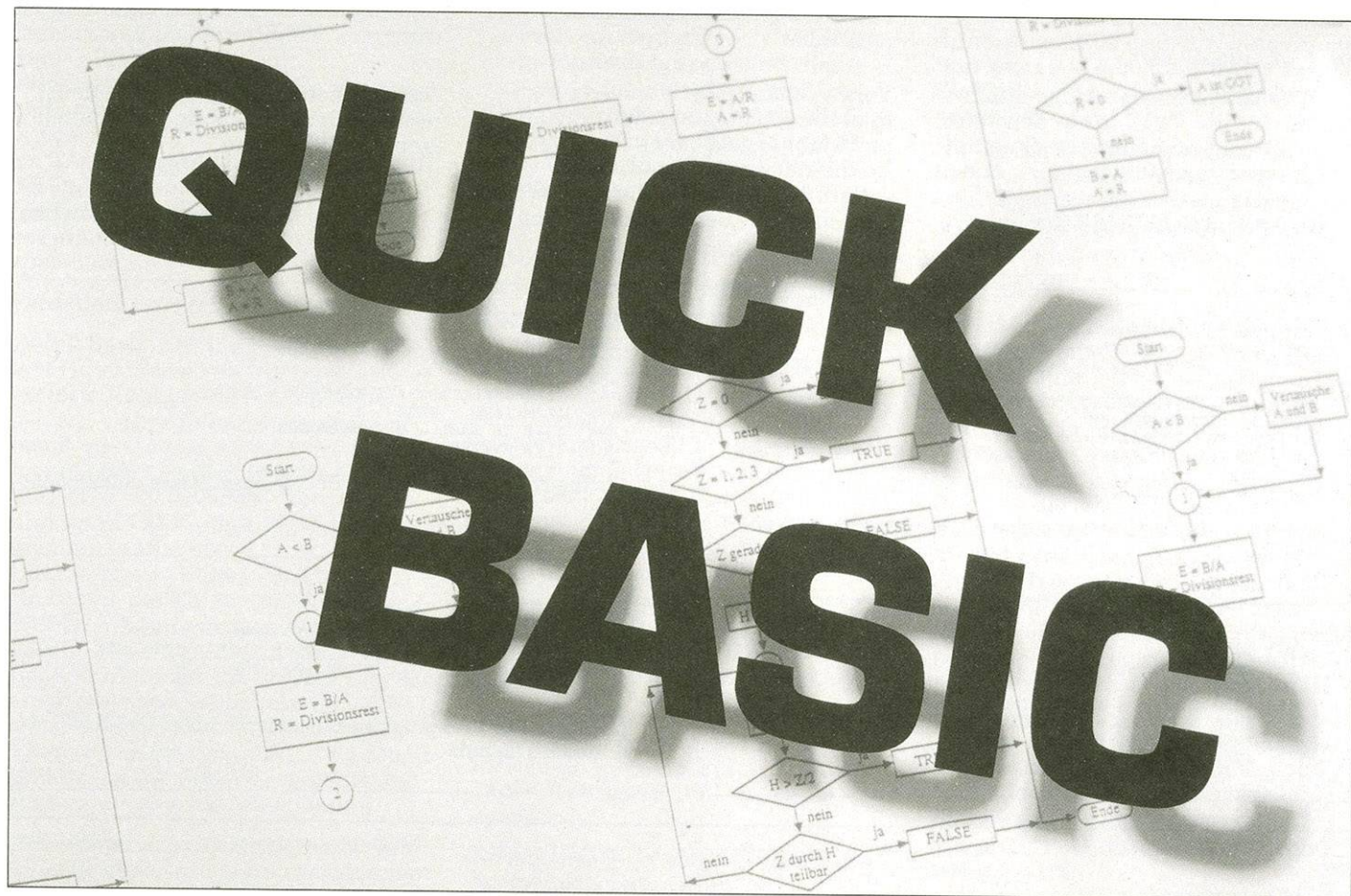
Fachredakteur/in

Sie haben bereits grundlegende Erfahrungen im journalistischen Bereich, fundierte PC- und Softwarekenntnisse, Sie drücken sich gewandt aus und bringen die Bereitschaft mit, selbständig zu arbeiten.

Sind Sie interessiert? Dann senden Sie uns Ihre Bewerbungsunterlagen. Wir freuen uns auf ein persönliches Gespräch.



Redaktionsbüro Everts & Hagedorn
Redaktion DOS International
Hans-Stießberger-Str. 3
8013 Haar/München



Strukturierte Programmierung mit Quick Basic 4.0

Basic-Programmierer werden von »EDV-Freaks«, die in Pascal, C oder gar Assembler programmieren, oft belächelt, weil Basic automatisch mit »Spaghetti-Code« und chaotischer Programmierung gleichgesetzt wird. Dabei haben die neuen Compiler Turbo Basic und Quick Basic längst eine neue »Basic-Ära« eingeläutet.

Die Zeiten, als man noch mit Zeilennummern programmierte und Prozeduren und Overlays nur bei anderen Sprachen möglich waren, liegen, seit es Turbo Basic und Quick Basic gibt, lange zurück. Das Beispiel des kleinen Computerspiels »Superhirn« zeigt auf, welche Wege Ihnen mit Quick Basic 4.0 offenstehen und wie Sie diese sinnvoll einsetzen. Sie bekommen – neben der Information über strukturierte Programmierung – ein Computerspiel, das Ihnen einige Nüsse zu knacken gibt.

Eine wichtige Komponente jeder Software stellt die Programmumgebung dar. Ein Produkt, das professionellen Ansprüchen genügen soll, muß folgende Bedingungen erfüllen:

- Die Texte in den Masken müssen farbig oder durch Hell-Dunkel-Kontraste abgesetzt sein.
- Die Programmbedienung erfolgt durch das Betätigen weniger Tasten (keine Input-Anweisungen).
- Die verschiedenen Bildschirmabschnitte müssen gelöscht werden können,

um die Benutzeroberfläche übersichtlicher zu gestalten.

Es bietet sich daher an, ein Programm zu schreiben, das Sie immer wieder verwenden können, das heißt, eine allgemeine Programmumgebung, die sinnvolle und leistungsfähige Tools enthält. Die Programmumgebung können Sie bei Quick Basic 4.0 nicht mehr mit Hilfe einer Include-Datei einbinden, weil Version 4.0 keine Prozeduren in Include-Dateien erlaubt. Dennoch gibt es eine simple Lösung für dieses Problem. Die Programmumgebung wird einfach als Teilprogramm geladen.

Die Zeiten, in denen man mit Zeilennummern programmierte, sind für Quick Basic schon längst vorbei

Eine wichtige Frage lautet deshalb: Was soll die Programmumgebung leisten? Hier kommen die Prozeduren ins Spiel. So ist ein Programm wesentlich transparenter, wenn man mit Proze-

durnamen arbeitet, die genau Auskunft darüber geben, was im Programm ausgeführt wird.

Abzuraten ist von Beispielen, wie in Listing 1 vorgestellt. Das »typische« Basic-Programm ist nämlich sehr undurchsichtig. Viele Fragen (Was macht das Programm in Zeile 10? Was beinhaltet das Unterprogramm ab Zeile 2000?) bleiben offen. Daher finden Sie hier eine Anleitung, wie man Prozeduren einsetzt. Aus diesen besteht nämlich im wesentlichen das Programm. Das Hauptprogramm dagegen ist nur wenige Zeilen lang.

Im Gegensatz zu Unterprogrammen mit Anweisungen wie »gosub« und »return«, sind Prozeduren selbständige Teilprogramme. Sie rufen sie mit

```
call Prozedurname(Parameter-
liste)
```

auf.

Eine Prozedur öffnen Sie mit der Anweisung

```
»sub Prozedurname(Parameter-
liste)«
```

Die Parameter müssen typen-

gleich, aber nicht gleichnamig sein. Die Inhalte übergeben Sie wie im Beispiel (siehe Listing 2).

Die Klammern kennzeichnen ein Array oder einen Bereich. Eine Prozedur können Sie zum Beispiel über ein Abbruchkriterium mit »exit sub« verlassen. Den Befehl »end sub« setzen Sie automatisch an das Ende der Prozedur. Quick Basic schiebt jede Prozedur in eine separate Bildschirmenebene. Dies hat den Vorteil, daß eine Prozedur – wenn irgend möglich – nur eine Bildschirmseite lang ist. Die Programme sind dadurch wesentlich übersichtlicher.

Es empfiehlt sich, diese Module separat zu testen, was bedeutet, daß Sie Fehler im Programm wesentlich früher und einfacher finden. Die Programme lassen sich dadurch leichter pflegen und aktualisieren. Mit der Funktionstaste [F2] blenden Sie eine Prozedurliste ein. Eine Prozedur können Sie auch ohne die Anweisung »call« aufrufen. Bei der Parameterübergabe fallen dann die Klammern weg. Die in der Programmumgebung enthaltenen Tools sehen Sie in Tabelle 1.

Durch Prozeduren können Sie Ihren Programmen eine modulare Struktur geben

Vollziehen Sie nun am Beispiel des Rahmens den Aufbau einer Prozedur nach. Die Prozedur übernimmt die Parameter »x1«, »y1«, »x2« und »y2« und zeichnet einen Rahmen.

Sie übergeben die Daten mit der Anweisung

```
call Rahmen(20,10,60,14)
```

Den Aufbau einer Prozedur zeigt Listing 3.

Es ist sinnvoll, die Parameter »x« und »y« als ganzzahlige Werte (Integer) zu definieren, um eine schnelle Bildschirmsteuerung zu erzielen.

Ein weiteres interessantes Modul ist »laufschrift«. An die Prozedur »laufschrift« übergeben Sie einen Text und eine Zahl, die bestimmt, wie oft die Schrift laufen soll. Dazu stellen Sie die Länge des Textes fest und zeichnen einen Rahmen, der sich in der Mitte des Bildschirms befindet. Die Größe des Rahmens ist natürlich ab-

hängig von der Länge des Lauftextes. Bevor Sie den Rahmen zeichnen, müssen Sie den entsprechenden Bildschirmausschnitt löschen. Die Prozedur soll so geartet sein, daß man über Tastendruck die Laufschrift stoppen kann. Weil die Laufschrift auf allen Systemen in nahezu gleicher Geschwindigkeit ablaufen soll, bauen Sie eine Warteschleife ein, die die Funktion »timer« verwendet (Listing 4).

Die Prozedur »laufschrift« übergibt die Konstante »titel\$« an die Variable »text\$«. Konstanten können jeden Datentyp haben, und Sie dürfen sie im Gegensatz zu Variablen nicht mehr ändern. Eine Zuweisung wie

```
titel$="NEU"
```

ist also ungültig.

Die Variablen »x1« und »x2« berechnen Sie, um den Rahmen zu zeichnen, wobei der Rahmen bei der x1-Koordinate (Bildmitte[40] minus halbe Länge des Textes minus 1) beginnt und bei x2 endet. Natürlich ist es möglich, das Modul »Rahmen« aus der Prozedur »laufschrift« aufzurufen und für die Problemlösung zu verwenden. Das Ausdrucken der Schrift verzögern Sie über eine Warteschleife.

Nun sind Sie in der Lage, die komplette Programmumgebung zu realisieren (Listing 5).

Es ist empfehlenswert, nützliche Routinen in Bibliotheken einzubinden

Sie können das Modul als Quick-Basic-Bibliothek verwenden oder als Programmumgebung über den Aufruf »Datei laden« in das Programm einbinden. Es bietet sich an, ein Tool in eine Bibliothek zu übernehmen, wenn die Prozedur zum Beispiel anderssprachig ist (C, Pascal oder Assembler) oder wenn Sie den Sprachschatz erweitern wollen. Somit wird der Compiler an Ihre besonderen Ansprüchen angepaßt. Die Bibliothek starten Sie mit

```
qb /l bibnam.q1b
```

Ein weiteres Hilfsmittel für die Programmierung sind die Include-Dateien. Include-Dateien sind Programmteile, die der Compiler während des Compilierens einbindet. Sie haben den Vorteil, daß

Name	Beschreibung
hell	Bildschirm auf invers umschalten
normal	Zurückschalten auf Hintergrund dunkel, Zeichen hell
intensiv	Zeichen heller
blinkt	Zeichen blinken
delay 2	Wartet 2 Sekunden
klein.raahmen	Zeichnet einen doppelten Rahmen mit den Koordinaten »x1«, »y1«, »x2« und »y2«.
rahmen	Wie oben, jedoch mit einfachen Rahmen
loesche	Löscht einen Bildschirmteil, um einen Rahmen zeichnen zu können.
taste	Nimmt eine Zeichenkette mit der Länge eins auf.
weiter	Schreibt in die Zeile 24, Spalte 1 »weiter nach Tastendruck«.
laufschrift	Zeichnet einen Rahmen. Im Rahmen läuft ein Text.

Tabelle 1. Die verschiedenen Werkzeuge

```
[SR1] 1: 10 COLOR 8,15
[CU2] 2: 20 GOSUB 1000 'menu
[LG5] 3: 30 PRINT "Weiter nach Tastendruck"
[F24] 4: 40 a$=INKEY$ : IF a$="" THEN GOTO 40
[MS1] 5: 50 GOSUB 2000
[NM1] 6: 60 COLOR 7,1
```

Listing 1. Unübersichtliche Programme wie in »schlecht.bas« sollten Sie vermeiden

```
[PG4] 1: SUB Uprol(Namen$, Zahl(), n)
[G02] 2:
[SN2] 3:
[885] 4: CALL Uprol(Text$, Bereich(), ENDE)
```

Listing 2. Wie Sie die einzelnen Module aufbauen, sehen Sie in »prozedur.bas«

```
[TR2] 1: 'Datei:RAHMEN.BAS
[PN0] 2: CLS
[4S3] 3: CALL Rahmen(20, 10, 60, 14)
[3S4] 4: '----- Konstanten
[E20] 5: '
[4M0] 6: END
[WC5] 7: '----- ENDE Hauptprogramm -----
[000] 8:
[000] 9:
[2T4] 10: '----- Modul Rahmen -----
[DM1] 11: DEFINT X-Y
[7Z3] 12: SUB Rahmen (x1, y1, x2, y2)
[RX3] 13: LOCATE y1, x1: PRINT "r"
[1W3] 14: LOCATE y1, x2: PRINT "l"
[NX3] 15: LOCATE y2, x1: PRINT "l"
[7W3] 16: LOCATE y2, x2: PRINT "r"
[ZX6] 17: LOCATE y1, x1 + 1: PRINT STRING$(x2 - x1 - 1, "=")
[1Y6] 18: LOCATE y2, x1 + 1: PRINT STRING$(x2 - x1 - 1, "=")
[4S5] 19: ' zeichnet x2-x1-1 mal das Zeichen ----^
[000] 20:
[A53] 21: FOR y = y1 + 1 TO y2 - 1
[RX3] 22: LOCATE y, x1: PRINT "l"
[SK3] 23: LOCATE y, x2: PRINT "r"
[ZS0] 24: NEXT
[HA1] 25: END SUB
```

Listing 3. Der Aufbau der Prozedur »rahmen1.bas«


```

[DK2] 1: 'Datei: LAUF.BAS
[PNO] 2: CLS
[DW9] 3: CONST Titel$ = "QuickBASIC ist spitze* Quic
      kBASIC ist spitze"
[EB4] 4: CALL LaufSchrift(Titel$, 2)
[WE3] 5: ' 2 Durchgänge -----
[4M0] 6: END
[000] 7:
[HK1] 8: DEFINT I-J
[C81] 9: DEFSNG Y
[0A5] 10: SUB LaufSchrift (Text$, Anzahl)
[000] 11:
[DY2] 12: Laenge = LEN(Text$)
[TK8] 13: x1 = INT(40 - Laenge / 2) - 1 'x-Koordina
      te für Lauftext
[CN3] 14: x2 = 40 + INT(Laenge / 2) + 1
[D24] 15: CALL Loesche(x1, 11, x2, 13)
[XW3] 16: CALL Rahmen(x1, 11, x2, 13)
[F74] 17: LOCATE 12, x1 + 1: PRINT Text$
[933] 18: FOR j = 1 TO Anzahl
[U13] 19:   FOR i = 1 TO Laenge
[637] 20:     Druck$ = MID$(Text$, i, Laenge) + LEFT$(
      Text$, i)
[EL1] 21:     a$ = INKEY$
[GG3] 22:     IF a$ <> "" THEN EXIT SUB
[795] 23:     '----- Abbruch bei Tastendruck
[LE4] 24:     LOCATE 12, x1 + 1: PRINT Druck$
[461] 25:     Delay .2
[1H4] 26:     '----- wartet 0,2 Sekunden
[YC1] 27:   NEXT i, j
[HA1] 28: END SUB
[000] 29:
[6A4] 30: SUB Loesche (x1, y1, x2, y2)
[MD2] 31:   FOR y = y1 TO y2
[SB6] 32:     LOCATE y, x1: PRINT STRING$(x2 - x1 + 1,
      " ")
[ZS0] 33:   NEXT
[HA1] 34: END SUB
[000] 35:
[2L2] 36: SUB Delay (Zeit)
[3Y2] 37:   Ende = TIMER + Zeit
[NB0] 38:   DO
[0U3] 39:   LOOP UNTIL TIMER > Ende
[HA1] 40: END SUB

```

Listing 4. Die Version »lauf1.bas«

```

[H63] 1: DECLARE SUB Menue ()
[SE3] 2: DECLARE SUB Taste (x$)
[RC3] 3: DECLARE SUB Weiter ()
[Z23] 4: DECLARE SUB Hell ()
[W53] 5: DECLARE SUB Hilfe ()
[4N6] 6: DECLARE SUB LaufSchrift (Text$, Anzahl!)
[4X3] 7: DECLARE SUB intensiv ()
[EC3] 8: DECLARE SUB blinkt ()
[6D3] 9: DECLARE SUB normal ()
[RN5] 10: DECLARE SUB Rahmen (x1%, y1%, x2%, y2%)
[2W5] 11: DECLARE SUB Loesche (x1%, y1%, x2%, y2%)
[414] 12: DECLARE SUB Delay (Zeit!)
[000] 13:
[743] 14: 'Datei : PROG-UM.BAS
[M82] 15: 'Stand : 01.12.88
[ZD2] 16: 'Update: 03.12.88
[AV3] 17: 'Autor : Gerd Kebschull
[E20] 18:
[H64] 19: ' C by DOS-International
[000] 20:
[082] 21: '*****
[6V1] 22: '***
[PT5] 23: '*** Allgemeine Programmumgebung ***
[6V1] 24: '***
[082] 25: '*****
[000] 26:
[TT1] 27: SUB blinkt
[CL1] 28:   COLOR 16, 1
[HA1] 29: END SUB
[000] 30:
[2L2] 31: SUB Delay (Zeit)
[3Y2] 32:   Ende = TIMER + Zeit
[NB0] 33:   DO
[0U3] 34:   LOOP UNTIL TIMER > Ende
[HA1] 35: END SUB
[000] 36:
[8D1] 37: SUB Hell
[9N1] 38:   COLOR 8, 15
[HA1] 39: END SUB
[000] 40:
[D72] 41: SUB intensiv

```

```

[2L1] 42: COLOR 10, 0
[HA1] 43: END SUB
[000] 44:
[DM1] 45: DEFINT X-Y
[C55] 46: SUB klein.Rahmen (x1, y1, x2, y2)
[4X5] 47:   LOCATE y1, x1: PRINT "┌" 'Alt> 218
[5T5] 48:   LOCATE y1, x2: PRINT "┐" 'Alt> 191
[AT5] 49:   LOCATE y2, x1: PRINT "└" 'Alt> 192
[SV5] 50:   LOCATE y2, x2: PRINT "┘" 'Alt> 217
[FE8] 51:   LOCATE y1, x1 + 1: PRINT STRING$(x2 - x1 -
      1, "-") 'Alt> 196
[SX6] 52:   LOCATE y2, x1 + 1: PRINT STRING$(x2 - x1 -
      1, "-")
[000] 53:
[A53] 54:   FOR y = y1 + 1 TO y2 - 1
[2V5] 55:     LOCATE y, x1: PRINT "│" 'Alt> 179
[3X3] 56:     LOCATE y, x2: PRINT "│"
[ZS0] 57:   NEXT
[HA1] 58: END SUB
[000] 59:
[HK1] 60: DEFINT I-J
[C81] 61: DEFSNG Y
[0A5] 62: SUB LaufSchrift (Text$, Anzahl)
[000] 63:
[DY2] 64:   Laenge = LEN(Text$)
[6P8] 65:   x = INT(40 - Laenge / 2) - 1 'x-Koordinat
      e für Lauftext
[CN3] 66:   x2 = 40 + INT(Laenge / 2) + 1
[0Z3] 67:   CALL Loesche(x, 11, x2, 13)
[DH3] 68:   CALL Rahmen(x, 11, x2, 13)
[H74] 69:   LOCATE 12, x + 1: PRINT Text$
[933] 70:   FOR j = 1 TO Anzahl
[U13] 71:     FOR i = 1 TO Laenge
[637] 72:       Druck$ = MID$(Text$, i, Laenge) + LEFT$(
      Text$, i)
[EL1] 73:       a$ = INKEY$
[GG3] 74:       IF a$ <> "" THEN EXIT SUB
[795] 75:       '----- Abbruch bei Tastendruck
[NC4] 76:       LOCATE 12, x + 1: PRINT Druck$
[461] 77:       Delay .2
[YC1] 78:     NEXT i, j
[HA1] 79:   END SUB
[000] 80:
[991] 81: DEFINT Y
[6A4] 82: SUB Loesche (x1, y1, x2, y2)
[MD2] 83:   FOR y = y1 TO y2
[SB6] 84:     LOCATE y, x1: PRINT STRING$(x2 - x1 + 1,
      " ")
[ZS0] 85:   NEXT
[HA1] 86: END SUB
[000] 87:
[GL1] 88: DEFSNG X-Y
[HU1] 89: SUB normal
[HE1] 90:   COLOR 7, 1
[HA1] 91: END SUB
[000] 92:
[DM1] 93: DEFINT X-Y
[723] 94: SUB Rahmen (x1, y1, x2, y2)
[R55] 95:   LOCATE y1, x1: PRINT "┌" 'Alt> 201
[VH5] 96:   LOCATE y1, x2: PRINT "┐" 'Alt> 187
[BK5] 97:   LOCATE y2, x1: PRINT "└" 'Alt> 200
[4K5] 98:   LOCATE y2, x2: PRINT "┘" 'Alt> 188
[ZX6] 99:   LOCATE y1, x1 + 1: PRINT STRING$(x2 - x1 -
      1, "-")
[XR2] 100:   '-----
[1Y6] 101:   LOCATE y2, x1 + 1: PRINT STRING$(x2 - x1 -
      1, "-")
[000] 102:
[A53] 103:   FOR y = y1 + 1 TO y2 - 1
[1H5] 104:     LOCATE y, x1: PRINT "│" 'Alt> 186
[SK3] 105:     LOCATE y, x2: PRINT "│"
[ZS0] 106:   NEXT
[HA1] 107: END SUB
[000] 108:
[GL1] 109: DEFSNG X-Y
[X32] 110: SUB Taste (x$)
[NB0] 111:   DO
[WM1] 112:     x$ = INKEY$
[ET2] 113:   LOOP UNTIL x$ <> ""
[HA1] 114: END SUB
[000] 115:
[2U1] 116: SUB Weiter
[VP0] 117:   Hell
[7U7] 118:   LOCATE 24, 1: PRINT "... weiter nach Taste
      ndruck";
[TB3] 119:   PRINT STRING$(40, " ");
[NB0] 120:   DO
[EL1] 121:     a$ = INKEY$
[RS2] 122:   LOOP UNTIL a$ <> ""
[441] 123:   normal
[HA1] 124: END SUB

```

Listing 5. Die Programmumgebung »pro-um.bas«

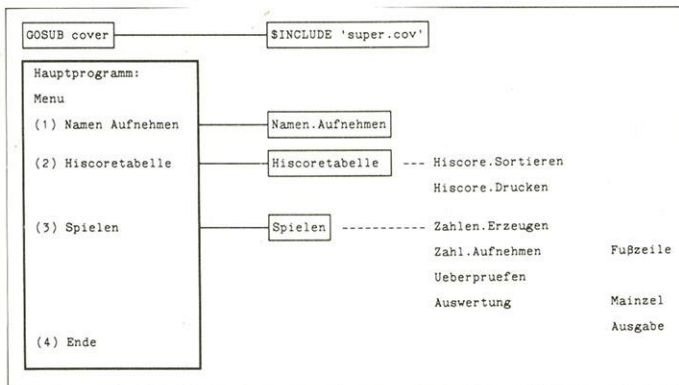


Bild 1. Die Programmübersicht zu Superhirn

sie im Programm nur eine Zeile einnehmen und das Programm somit übersichtlich bleibt. Da Quick Basic Prozeduren in separaten Ebenen unterbringt, ist es ab Version 4.0 nicht mehr möglich, Prozeduren in Include-Dateien einzubinden.

Include-Dateien verwendet man, um zum Beispiel Konstanten zu definieren oder Masken abzubilden. Die Syntax für Include-Dateien lautet:

```
rem $include: "Dateiname.  
Erweiterung"
```

Als erstes entwickeln Sie nun das Superhirn-Cover (Listing 6) und speichern es als Unterprogramm »cover.cov«. Das Titel- und Schlußbild rufen Sie anschließend mit

```
gosub cover
```

auf. Die Datei binden Sie am Ende des Hauptprogramms als Include-Datei ein.

```
end  
rem $include: 'super.cov'
```

Vor der Include-Datei befindet sich eine End-Anweisung (schützender End-Block), die verhindert, daß man vom Hauptprogramm in das Unterprogramm gelangt (»Fehlers Return ohne Gosub«).

Ein größeres Programm sollten Sie immer mit einem Programmkopf versehen. Denn nichts ist ärgerlicher, als ein Programm, das man nicht mehr einem bestimmten Listing zuordnen kann. Der Programmkopf beinhaltet folgende Daten:

Dateiname:
Stand:
Letzte Änderung:
Version:
Autor:
Software:
Programmbeschreibung:
Copyright-Vermerke:
Konstantendeklaration:

Einen Rahmen um die Daten gestalten Sie einfach mit den Grafikzeichen Ihres Computers. Diese erhalten Sie mit Hilfe der Alt-Taste und dem Ziffernblock auf den Bildschirm. Die obere linke Ecke eines einfachen Rahmens erzeugen Sie beispielsweise mit [Alt] und der Zahlenfolge [2], [1] und [8]. Nach Loslassen der Taste [Alt] steht das Zeichen auf dem Bildschirm. Den »Kopf« speichern Sie entweder unter dem Dateinamen »kopf.bas« oder unter »kopf.box«.

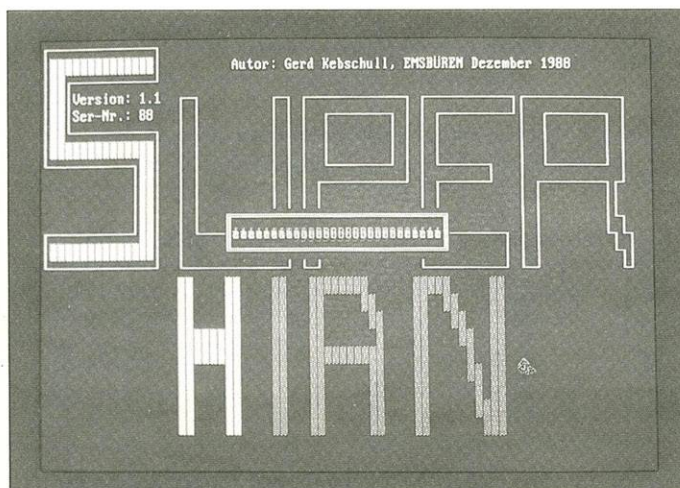


Bild 2. Das Eingangsменю von Superhirn

```
[231] 1: Cover:  
[WU8] 2: PRINT "  
[F48] 3: PRINT "  
[VSA] 4: PRINT "  
[UEW] 5: PRINT "  
[YFR] 6: PRINT "  
[D1K] 7: PRINT "  
[HFH] 8: PRINT "  
[L8V] 9: PRINT "  
[EES] 10: PRINT "  
[UPC] 11: PRINT "  
[FZC] 12: PRINT "  
[TET] 13: PRINT "  
[1TG] 14: PRINT "  
[PMV] 15: PRINT "  
[GHD] 16: PRINT "  
[G4C] 17: PRINT "  
[G4C] 18: PRINT "  
[4RC] 19: PRINT "  
[MUE] 20: PRINT "  
[GKB] 21: PRINT "  
[ØEB] 22: PRINT "  
[GVB] 23: PRINT "  
[GKB] 24: PRINT "  
[651] 25: RETURN
```

Listing 6. Die Eingangsmaske »super.cov«

Bevor Sie nun die Arbeit mit den Superhirn-Programm beginnen, müssen Sie sich über die Aufgabenstellung klar werden. Der Computer erzeugt eine vierstellige Zahl. Keine Ziffer darf in diesem Programm doppelt erzeugt werden. Der Spieler gibt ebenfalls eine vierstellige Zahl ein und bekommt eine Information über die Richtigkeit seiner Eingabe. So bedeutet beispielsweise

2.1 daß er zwei Ziffern an der richtigen Stelle getroffen hat. Eine dritte Ziffer ist zwar in der vom Computer erzeugten Zahl enthalten, aber an einer anderen Position. Nachdem der Programmkopf geladen ist und Sie die Programmumgebung fertiggestellt und als Teilprogramm eingebunden haben, bauen Sie die Struktur des Spiels auf. Natürlich kann man nun einfach »drauflos« programmieren. Sinnvoller und zeitsparender ist es aber, erst

einmal eine Übersicht zu entwickeln, aus der man den Programmfluß erkennt. Ein sinnvolles Mittel ist hier der Programmablaufsplan oder eine Programmübersicht wie in Bild 1.

Programmablaufsplan oder Programmübersicht ersparen Ihnen eine Menge Ärger und Zeit

Sie haben nun die Programmumgebung geschaffen, den Aufbau der Prozeduren kennengelernt, das Cover entwickelt (siehe Bild 2), einen Programmkopf gespeichert, die Superhirn-Aufgabe definiert und sich Gedanken über den Ablauf des Programms gemacht, das heißt, eine Programmübersicht skizziert. Nun entwickeln Sie das Programm Prozedur für Prozedur und testen es aus.


```

[SH1] 1: SUB hilfe
[PN0] 2: CLS
[093] 3: CALL Rahmen(1, 1, 80, 4)
[VP0] 4: hell
[P76] 5: LOCATE 2, 25: PRINT "      S U P E R H I R
      N
[XY7] 6: LOCATE 3, 25: PRINT "      ~~~~~~
[WC2] 7: PRINT : normal
[D8A] 8: PRINT "Der Rechner hat eine vierstellige Z
ufallszahl erzeugt."
[VY7] 9: PRINT "Es wird keine Ziffer doppelt vorkom
men."
[CV9] 10: PRINT "Der Spieler gibt ebenfalls eine vie
rstellige Zahl ein"
[U37] 11: PRINT "und bekommt folgende Informationen:
"
[VP0] 12: hell
[CT4] 13: PRINT : PRINT "      2.3
[441] 14: normal
[V5B] 15: PRINT "Dies bedeutet, er hat insgesamt dre
i Ziffern richtig, wobei "
[R47] 16: PRINT "zwei an der richtigen Stelle stehen
"
[AR2] 17: PRINT "Beispiel:"
[0H1] 18: intensiv
[GL3] 19: PRINT : PRINT "      1 3 6 4"
[VP0] 20: hell
[GP3] 21: PRINT : PRINT "      2.3 "
[441] 22: normal
[PH4] 23: CALL klein.Rahmen(1, 19, 80, 22)
[80B] 24: LOCATE 20, 2: PRINT Ausrichten("Lauftexte
und Töne können durch", 1)
[XHC] 25: LOCATE 21, 2: PRINT Ausrichten("Drücken ei
ner Taste abgebrochen werden", 1)
[N31] 26: Weiter
[HA1] 27: END SUB

```

Listing 11. Die Hilfeprozedur »hilfe.bas«

```

[SZ5] 1: FUNCTION Ausrichten$(Text$, minus)
[CN5] 2: rest = INT((80 - LEN(Text$)) / 2) - minus
[HY3] 3: 'zieht minus-Zeichen ab
[2V8] 4: Ausrichten$ = STRING$(rest, " ") + Text$ +
STRING$(rest, " ")
[R52] 5: END FUNCTION

```

Listing 12. Mit »ausricht.bas« zentrieren Sie den Text

```

[DM1] 1: DEFINT X-Y
[1M5] 2: SUB NamenAufnehmen (Spielername$)
[PN0] 3: CLS
[6M0] 4: x = 1
[272] 5: FOR y = 1 TO 11
[GP5] 6: CALL klein.Rahmen(x, y, 80 - x, 24 - y)
[231] 7: x = x + 1
[2S0] 8: NEXT
[VM8] 9: Text$ = "*" Bitte Namen eingeben * Bitte Na
men eingeben"
[234] 10: CALL LaufSchrift(Text$, 1)
[7W6] 11: LOCATE 12, 17: PRINT STRING$(LEN(Text$) +
1, " ")
[235] 12: LOCATE 12, 17: INPUT Spielername$
[HA1] 13: END SUB

```

Listing 13. Mit »name.bas« übernehmen Sie den Spielernamen

Die Funktion wählen Sie mit [F2] und den Cursor-Tasten an, die Option »Bewegen« mit [Tab]. Nach Umschalten auf »prog-um.bas« übergeben Sie die Funktion an die Programmumgebung.

Wenn Sie eine Hiscore-Tabelle in das Programm aufnehmen wollen, müssen Sie nun den Namen des Spielers abfragen. Die Möglichkeit,

den Namen zu wechseln, müssen Sie in das Hauptmenü aufnehmen, da ein anderer Spieler ebenfalls versuchen wird, das Superhirn zu knacken. Nehmen Sie den Namen nicht auf mit

```
cls
input"Geben Sie Ihren Namen an";Spieler$
```

Es sollen vielmehr die Module »klein.raahmen« und »lauf-

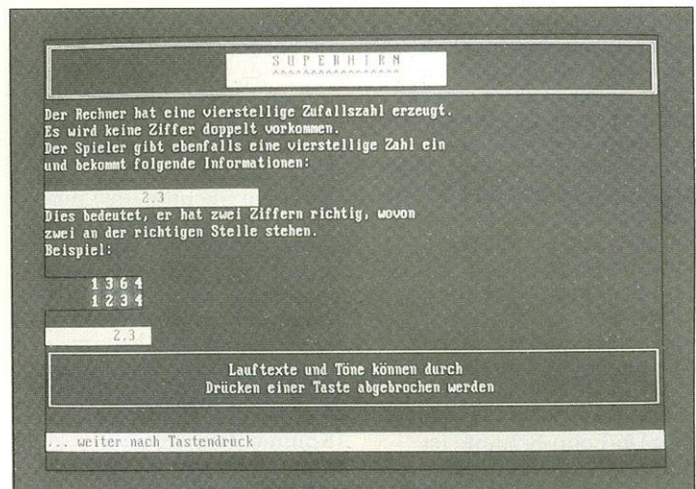


Bild 3. Bei Problemen schalten Sie um zum Hilfsbildschirm

schrift« zum Einsatz kommen, die die Angelegenheit interessanter gestalten (Listing 13).

Das Teilprogramm testen Sie vom Hauptprogramm aus. Mit »end« verhindern Sie, daß Sie die schon fertigen Programmsegmente aus-

```
drucken
call NamenAufnehmen
(Spielername$)
end
```

Nun müssen Sie sich Gedanken über das Programmenü machen. Das Menü ist die Schaltzentrale des Programms. Von hier aus rufen Sie sämtliche Teilprogramme (also Prozeduren) auf:

- (1) Name des Spielers aufnehmen (bei Spielerwechsel)
- (2) Hiscore-Tabelle ausdrucken
- (3) Superhirn spielen
- (4) Ende

Sinnvoll ist es auch, dafür zu sorgen, daß Sie das Hilfsmenü (Bild 3) aus dem Hauptmenü anwählen können, um beispielsweise bei einem Spielerwechsel das Spiel zu erklären (Listing 14).

Das Hauptmenü ist die Schaltzentrale eines Programms und verbindet die einzelnen Teilprogramme miteinander

In dieser Routine haben Sie einmal nicht mit der Prozedur »rahmen« gearbeitet – denn zuviel des Guten ist auch langweilig. Aber wie Sie sehen, erzielt man auch mit den Prozeduren »normal«, »hell«, »intensiv« und einer vernünftigen Bildschirmteilung gute Ergebnisse. Das Umleiten in die einzelnen Prozeduren erfolgt über die Anweisung »select case«.

Diese Anweisung eignet sich besonders gut, einen Parameter zu untersuchen. Ist die Bedingung erfüllt, führt das Programm die nachfolgenden Anweisungen aus. Hier die Syntax:

```
select case variable
case bedingung1
anweisung1
anweisung2
case bedingung2
anweisung3
anweisung4
case else
anweisung5
end select
```

Mit der Anweisung »case else« vermeiden Sie eine Fehlbedienung. Sie wird nur dann ausgeführt, wenn keine Case-Bedingung erfüllt wurde. In Listing 15 sehen Sie ein Beispiel für das Programm »superhirn«.

Wenden Sie sich nun dem wichtigsten Programmteil zu – der Prozedur »spielen«, aus der Sie die Prozeduren »zahlen.erzeugen«, »zahl.aufnahme«, »ueberprüfen« und »auswertung« aufrufen (Listing 16). An die Prozedur »spielen« übergeben Sie die Bereiche »rechner()« und »spieler()« sowie die Parameter »max« und die Anzahl der Versuche. Aus dieser Prozedur rufen Sie folgende Teilprogramme auf:

```
zahlen.erzeugen: Erzeugen der
Superhirn-Zahl
zahl.aufnehmen: Aufnahme der
Spielerzahl
ueberprüfen: Vergleich der
Superhirn-Zahl mit der
Spielerzahl
auswertung: Auswertung muß am
Ende erfolgen
```

Die Prozedur »fusszeile« (Listing 17) benötigen Sie, wenn Sie die Fehlermeldungen in der Zeile 24 ausdrucken wollen. Sie liefert also eine Spielanweisung.

Die Prozedur »zahlen.erzeugen« erzeugt die Superhirn-Zahl. Da vereinbart ist, daß keine Ziffer doppelt erscheinen darf, muß das Programm eine neue Zahl erzeugen, wenn diese Bedingung nicht erfüllt ist. Im Beispiel ist das Problem über eine Do-Loop-Schleife und eine Boolean-Anweisung gelöst.

Eine Boolean-Anweisung verwenden Sie wie ein Signal. Man kann daher den Wahrheitsgehalt über Abfragen feststellen wie

```
if signal1 = -1 then print ...
```

Eine Do-Loop-Schleife ist eine Endlos-Schleife, die man nur über einen Sprung zu einem Label verlassen kann, das sich außerhalb der Schleife befindet, oder mit der Anweisung »exit do« (siehe Listing 18).

Kommentare wirken sich bei Compilern nicht mehr negativ auf die Ausführungsgeschwindigkeit aus

Eine Do-Loop-Until-Schleife führt eine Schleife so lange fort, bis eine bestimmte Bedingung erfüllt ist. In diesem Zusammenhang ist auch die While-Anweisung gültig (Listing 19). Das Argument steht für einen numerischen Ausdruck, den Sie daraufhin überprüfen, ob er falsch (»false = 0«) oder wahr (»true < > 0«) ist.

»nocheinmal« ist ein Boolescher Ausdruck, den man verwendet, um den Wahrheitsgehalt zu überprüfen (siehe Listing 20). Damit verhindern Sie, daß man, wie in Basic allgemein üblich, mit »wildem« Goto-Springen hin- und herschaltet. Ihnen ist sicher aufgefallen, daß bislang im Programm noch überhaupt keine Goto-Anweisung aufgetaucht ist.

Es ist übrigens sinnvoll, die Super-Zahl für den Programmtest auszudrucken, weil Sie überprüfen müssen, ob sich ein Fehler im Programm befindet.

Da die Forderung gestellt ist, die Zahlen nicht über einfache Input-Anweisungen aufzunehmen, wird es jetzt etwas komplizierter. Die Aufgabe war, die Zahlen über die Funktion »inkey\$« aufzunehmen. Des weiteren muß eine falsche Eingabe über eine Taste gelöscht werden können.

Die Zahlen bestätigen Sie über [Enter] oder jede andere Taste, wenn alle Zahlen ein-

gegeben sind und Sie nicht die Löschtaaste betätigt haben. Hier verwaltet das Programm die Bildschirmkoordinaten über die Anweisungen »csrlin« und »pos(0)« (Listing 21).

Beachten Sie bitte, daß man durch die Verwendung von Anweisungen der Form »elseif« gänzlich auf Labels verzichten kann. Ebenso fällt Ihnen sicher auf, daß fast jede Programmzeile kommentiert ist. Da der Compiler beim Übersetzen des Programms die »Remarks« (Kommentare) »heraus-schneidet«, werden gut dokumentierte Programme nicht mehr mit einem langsameren Code bestraft.

Nun wird auch verständlich, warum in der Prozedur »spielen« die Prozedur »fusszeile« verwendet wird. Die Fehlermeldung »Fehler -> keine Bedingung erfüllt« wird in Zeile 24 ausgedruckt. Wenn Ihnen die einzelnen Abschnitte noch nicht ganz klar sind, ist es empfehlenswert, diese Prozedur im Einzelschrittmodus zu untersuchen und sich die Inhalte der Variablen ausgeben zu lassen.

Dazu lassen Sie sich mit der Tastenkombination [Alt-D Z] die Inhalte der Variablen »i«, »x«, »y« anzeigen und setzen mit [F9] sogenannte Haltepunkte (Breakpoints). Nachdem Sie das Programm mit [F5] gestartet und den ersten Haltepunkt erreicht haben, können Sie mit [F10] jeden einzelnen Schritt verfolgen.

»Pariert« Ihr Programm nicht, steht Ihnen ein leistungsfähiger Debugger zur Fehlersuche zur Verfügung

Nun wird deutlich, warum man zum Beispiel »x« auf 10 setzen muß, wenn der Cursor sich an der Stelle »x = 6« befindet und Sie die Löschtaaste (»#«) gedrückt haben.

Nachdem die Zahl aufgenommen ist, muß sie überprüft werden, damit Sie die Information ausgeben können (siehe Listing 22).

Nun müssen Sie noch das Modul »spielen« fertigstellen (Listing 23). Es ist wichtig, die Information zu übergeben, wie viele Versuche der Spieler benötigt. Den Parameter »versuch« teilen Sie nur deshalb der Prozedur »spielen« mit, damit man den Wert am Ende der Proze-

```
[DK1] 1: SUB Menue
[PN0] 2: CLS
[HP5] 3: LOCATE 7, 25: PRINT "Namen aufnehmen"
[ZU5] 4: LOCATE 9, 25: PRINT "Hiscore-Tabelle"
[DX4] 5: LOCATE 11, 25: PRINT "Superhirn"
[S14] 6: LOCATE 13, 25: PRINT "Hilfe"
[344] 7: LOCATE 15, 25: PRINT "E N D E"
[ØH1] 8: intensiv
[ZB5] 9: LOCATE 17, 25: PRINT "... wählen Sie"
[VPØ] 10: hell
[FM6] 11: LOCATE 5, 20: PRINT "          M E N
           U E
[WA3] 12: LOCATE 7, 20: PRINT " 1 "
[ZA3] 13: LOCATE 9, 20: PRINT " 2 "
[UB3] 14: LOCATE 11, 20: PRINT " 3 "
[ZB3] 15: LOCATE 13, 20: PRINT " 4 "
[4C3] 16: LOCATE 15, 20: PRINT " 5 "
[441] 17: normal
[HA1] 18: END SUB
```

Listing 14. Mit »menu1.bas« geben Sie dem Spieler Auswahlmöglichkeiten

```
[631] 1: CASE "1"
[RM5] 2: CALL NamenAufnehmen(SpielerName$)
[A31] 3: CASE "2"
[795] 4: CALL Hiscore.Drucken(Daten(), nr)
[E31] 5: CASE "3"
[2W7] 6: CALL Spielen(Rechner(), Spieler(), max,
           Versuche)
[K31] 7: CASE "4"
[BVØ] 8: hilfe
[P31] 9: CASE "5"
[9PØ] 10: Ende
[PF1] 11: CASE ELSE
[721] 12: Fehler
[GS1] 13: END SELECT
```

Listing 15. Sie können das Hauptprogramm auch aufbauen wie in »case.bas«

```
[GZ1] 1: DEFINT M, R-S
[8U7] 2: SUB Spielen (Rechner(), Spieler(), max, Ver-
           such)
[3V1] 3: Versuch = Ø
[PNØ] 4: CLS
[TE9] 5: PRINT "■ ■ ■ ■ ■": LOCATE 1, 60: PRINT "Datum
           m: "; Datum$(DATE$)
[P95] 6: LOCATE 2, 60: PRINT "Zeit
           "; TIME$
[ØØØ] 7: Fusszeile
[7N1] 8: Fusszeile
[P24] 9: LOCATE 3, 1: PRINT "Zahl:";
[8D4] 10: PRINT TAB(50); "Platz", "Zahl"
[SL1] 11: LOCATE 3, 1
[S16] 12: CALL Zahlen.Erzeugen(Rechner(), max)
[ØØØ] 13:
[WS5] 14: CALL Zahl.Aufnehmen(Spieler(), max)
[159] 15: CALL Ueberpruefen(Rechner(), Spieler(), ma-
           x, Zahl, Platz)
```

Listing 16. Die Routine »spiel.bas«

```
[MB2] 1: SUB Fusszeile
[VPØ] 2: hell
[3WA] 3: LOCATE 24, 1: PRINT "Zahl löschen mit '#'
           - bestätigen mit <Enter>";
[NB3] 4: PRINT STRING$(30, " ");
[441] 5: normal
[HA1] 6: END SUB
```

Listing 17. Die Fußzeile gestaltet »zeile.bas«

```
[NBØ] 1: DO
[8P1] 2: Anweisung
[L23] 3: IF Bedingung THEN EXIT DO
[HSØ] 4: LOOP
```

Listing 18. Den Aufbau der Schleife sehen Sie in »doloop.bas«


```
[H05] 1: DO [WHILE] [UNTIL] [Argument]
[W20] 2: .
[W20] 3: .
[FK1] 4: [EXIT DO]
[9A5] 5: LOOP [WEND] [WHILE] [UNTIL]
```

Listing 19. In »while.bas« sehen Sie die Struktur der While-Schleife

```
[XX1] 1: DEFINT I-J, N
[6A7] 2: SUB Zahlen.Erzeugen (Super() AS INTEGER, n)
[K32] 3: FOR i = 1 TO n
[YB3] 4: Super(i) = INT(RND * 9)
[ZS0] 5: NEXT
[ZL3] 6: 'Zahlen überprüfen
[GX0] 7: PRINT
[NB0] 8: DO
[U52] 9: nocheinmal = 0
[K32] 10: FOR i = 1 TO n
[KH2] 11: FOR j = i + 1 TO n
[U74] 12: IF Super(i) = Super(j) THEN
[NB2] 13: nocheinmal = -1
[2C3] 14: Super(j) = INT(RND * 9)
[801] 15: END IF
[ZC1] 16: NEXT j, i
[PF4] 17: LOOP UNTIL (nocheinmal <> -1)
[K32] 18: FOR i = 1 TO n
[9M2] 19: PRINT Super(i);
[AT2] 20: ' Programmtest--
[ZS0] 21: NEXT
[GX0] 22: PRINT
[HA1] 23: END SUB
```

Listing 20. Die Zahlen erzeugen Sie mit »zahl.bas«

```
[XX1] 1: DEFINT I-J, N
[167] 2: SUB Zahl.Aufnehmen (Spieler() AS INTEGER, n)
[6L0] 3: i = 0
[NB0] 4: DO
[NN1] 5: y = CSRLIN
[0G1] 6: x = POS(0)
[Z86] 7: CALL Taste(x$) ' gibt 1 Zeichen zurück
[000] 8:
[464] 9: IF x$ = "#" THEN ' Löschtaste
[BS5] 10: IF x = 6 THEN ' verhindert, daß x<10
[PS0] 11: x = 10
[TF6] 12: SOUND 37, 10 ' tiefer Ton -> falsche
Eingabe
[801] 13: END IF
[3C4] 14: LOCATE y, x - 2: PRINT "X";
[L15] 15: ' überschreibt ein Zeichen --
[4S6] 16: LOCATE y, x - 4 ' Cursor zurücksetzen
[VZ6] 17: IF i <> 0 THEN i = i - 1 'bei i=0-> Programmstart
[000] 18:
[FX5] 19: ELSEIF (i = max) AND (x$ = CHR$(13)) THEN
[RD4] 20: ' <Return>-Taste--
[YF2] 21: PRINT "<ENTER>";
[TE1] 22: EXIT SUB
[000] 23:
[K46] 24: ELSEIF ASC(x$) >= 48 AND ASC(x$) <= 57 THEN
H25:
[584] 25: ' +-----^-----
[584] 26: ' gültige Zeichen 1 bis 0
[611] 27: i = i + 1
[L87] 28: IF i = max + 1 THEN 'Abbruch nicht über
<Enter>
[YF2] 29: PRINT "<ENTER>";
[TE1] 30: EXIT SUB
[801] 31: END IF
[323] 32: Spieler(i) = VAL(x$)
[145] 33: LOCATE y, x + 1: PRINT Spieler(i);
[000] 34:
[726] 35: ELSEIF i = max THEN PRINT "<ENTER>": EXIT SUB
[2A5] 36: ' ^-- max-Zahlen aufgenommen
[000] 37:
[ZR0] 38: ELSE
[T16] 39: y = CSRLIN - 1 ' Y-Position speicher
n
[2U5] 40: x = POS(0) ' X-Position speicher
n
```

```
[VP0] 41: Hell
[EZ8] 42: LOCATE 24, 1: PRINT "Fehler -> Keine Bedingung erfüllt";
[HB3] 43: PRINT STRING$(20, " ");
[591] 44: heulen 2
[7N1] 45: Fusszeile
[AC6] 46: LOCATE y + 1, x ' Koordinaten übergeben
[801] 47: END IF
[VS2] 48: LOOP UNTIL (i > n)
[E45] 49: ' n+1. Zeichen Abbruchkrit.
[HA1] 50: END SUB
```

Listing 21. Die Zahlen nehmen Sie mit »zahlnehm.bas« auf

```
[6L2] 1: DEFINT I-J, N, R-S
[AV8] 2: SUB Ueberpruefen (Rechner(), Spieler(), n, Zahl, Platz)
[AW2] 3: Zahl = 0: Platz = 0
[K32] 4: FOR i = 1 TO n
[XC7] 5: IF Spieler(i) = Rechner(i) THEN Platz = Platz + 1
[L35] 6: ' Zahl und Platz richtig -----^
[M32] 7: FOR j = 1 TO n
[N27] 8: IF Spieler(i) = Rechner(j) THEN Zahl = Zahl + 1
[9U5] 9: 'nur Zahl richtig, Platz falsch --^
[ZC1] 10: NEXT j, i
[HA1] 11: END SUB
```

Listing 22. Die Überprüfung der Zahlen findet in »pruef.bas« statt

```
[8U7] 1: SUB Spielen (Rechner(), Spieler(), max, Versuch)
[3V1] 2: Versuch = 0
[PN0] 3: CLS
[TE9] 4: PRINT "#####": LOCATE 1, 60: PRINT "Datum:"; Datum$(DATE$)
[P95] 5: LOCATE 2, 60: PRINT "Zeit:"; TIME$
[000] 6: Fusszeile
[7N1] 7:
[P24] 8: LOCATE 3, 1: PRINT "Zahl:";
[8D4] 9: PRINT TAB(50); "Platz", "Zahl"
[SL1] 10: LOCATE 3, 1
[S16] 11: CALL Zahlen.Erzeugen(Rechner(), max)
[000] 12:
[NB0] 13: DO
[KC3] 14: Versuch = Versuch + 1
[323] 15: PRINT Versuch; ". ";
[NN1] 16: y = CSRLIN
[WS5] 17: CALL Zahl.Aufnehmen(Spieler(), max)
[159] 18: CALL Ueberpruefen(Rechner(), Spieler(), max, Zahl, Platz)
[8X4] 19: LOCATE y, 50: PRINT Platz, Zahl
[KZ2] 20: IF Platz = max THEN
[ZC4] 21: ' ^----- Lösung
[PL1] 22: LOCATE 1, 1
[VP0] 23: Hell
[BE2] 24: FOR i = 1 TO max
[RU2] 25: PRINT Rechner(i);
[ZS0] 26: NEXT
[441] 27: normal
[C31] 28: blinkt
[RN0] 29: BEEP
[LC6] 30: LOCATE 1, 20: PRINT "##### Lösung#####";
[251] 31: Delay 2
[N31] 32: Weiter
[K94] 33: CALL Auswertung(Versuch)
[TE1] 34: EXIT SUB
[801] 35: END IF
[K44] 36: LOOP UNTIL (Versuch >= 18)
[PT7] 37: 'IF Versuch >= 18 THEN CALL Auswertung(Versuch)
[HA1] 38: END SUB
```

Listing 23. Die erweiterte Prozedur »spiel2.bas«

dur auch an das Hauptprogramm zurückgeben kann. Im Modul »auswertung« kommt es darauf an, sehr gute Leistungen mit einem Lied zu belohnen. Hier ver-

wenden Sie wiederum die Module »rahmen« und »lauf-schrift«. Schön ist es, wenn die Bewertung (Listing 24) mit einer einheitlichen Aussage abschließt, beispiels-

weise »Sie haben 7 Versuche benötigt«.

Nun zu den Hilfsmodulen »ausgabe« (Listing 25), »mainzel« (Listing 26) und – nehmen Sie das nicht persönlich – »dubistverrückt« (Listing 27).

Das Superhirn-Spiel nimmt langsam Formen an. Sie müssen jetzt nur noch die Module »hiscore.drucken« und »ende« mit einer Meldung belegen, weil diese Prozeduren noch nicht installiert sind (Listing 28).

Superhirn verwendet die sequentielle Datei, um eine Hiscore-Tabelle auf dem Datenträger zu speichern. Doch bevor dieser Dateityp behandelt wird, ein kurzer Überblick über die Dateiararten, die Quick Basic erzeugt und verarbeitet. Quick Basic ist in der Lage, drei verschiedene Dateiararten zu verwalten:

- sequentielle Dateien
- wahlfreie Dateien (fälschlich auch oft als indexsequentielle Dateien bezeichnet)

```
[HK1] 1: DEFINT I-J
[184] 2: SUB Auswertung (Versuch)
[PN0] 3: CLS
[8Y2] 4: IF Versuch < 8 THEN
[C31] 5: blinkt
[XS3] 6: CALL Rahmen(19, 11, 60, 13)
[SK7] 7: LOCATE 12, 22: PRINT "**** S U P E R L E
I S T U N G ****"
[2C1] 8: Mainzel
[441] 9: normal
[GG3] 10: CALL Ausgabe(Versuch)
[801] 11: END IF
[000] 12:
[A36] 13: IF (Versuch >= 8) AND (Versuch <= 11) THEN
[G37] 14: Text$ = "* G U T E Leistung* G U T E L
eistung"
[192] 15: FOR y = 8 TO 16
[KX4] 16: LOCATE y, 1: PRINT Text$: Text$
[2S0] 17: NEXT
[234] 18: CALL LaufSchrift(Text$, 1)
[GG3] 19: CALL Ausgabe(Versuch)
[801] 20: END IF
[000] 21:
[L26] 22: IF (Versuch > 11) AND (Versuch <= 14) THEN
[Zy7] 23: LOCATE 12, 20: PRINT "**** befriedigende
LEISTUNG ****"
[251] 24: Delay 2
[GG3] 25: CALL Ausgabe(Versuch)
[801] 26: END IF
[000] 27:
[Zy5] 28: IF (Versuch > 14) AND (Versuch < 18) THEN
[C31] 29: blinkt
[MR8] 30: LOCATE 12, 20: PRINT "**** Das war nicht
besonders stark ****"
[543] 31: DuBistVerrueckt 2
[251] 32: Delay 2
[GG3] 33: CALL Ausgabe(Versuch)
[801] 34: END IF
[FG3] 35: IF (Versuch >= 18) THEN
[PN0] 36: CLS
[CD9] 37: LOCATE 11, 20: PRINT "Wir brechen hier
an dieser Stelle ab."
[D6A] 38: LOCATE 12, 20: PRINT "Bitte nicht RATEN s
ondern logisch denken"
[543] 39: DuBistVerrueckt 2
[251] 40: Delay 2
[GG3] 41: CALL Ausgabe(Versuch)
[N31] 42: Weiter
[801] 43: END IF
[HA1] 44: END SUB
```

Listing 24. Für die Bewertung des Spiels ist »auswert.bas« zuständig

```
[HK1] 1: DEFINT I-J
[0L3] 2: SUB Ausgabe (Versuch)
[AW3] 3: CALL Loesche(19, 11, 60, 13)
[0H1] 4: intensiv
[LH4] 5: CALL klein.Rahmen(19, 11, 60, 13)
[441] 6: normal
[HR9] 7: LOCATE 12, 24: PRINT " Sie haben "; Versuc
h; " Versuche benötigt"
[251] 8: Delay 2
[HA1] 9: END SUB
```

Listing 25. Die Datei »hilfAusg.bas«

```
[HK1] 1: DEFINT I-J
[F22] 2: SUB Mainzel
[726] 3: PLAY "o2 L8 c f g a a o3 c o2 a g a + a g L
4 f"
[NB3] 4: PLAY "p8 o3 l16 c d e f"
[HA1] 5: END SUB
```

Listing 26. Das kleine Soundprogramm »mainzel.bas«

```
[C64] 1: SUB DuBistVerrueckt (n)
[K32] 2: FOR i = 1 TO n
[425] 3: PLAY "o2 f4 p6 a+ p6 d8 f f f p4
"
[585] 4: a$ = INKEY$: IF a$ <> "" THEN EXIT SUB
[0K5] 5: ' Du bist verrückt mein Kind
[2S0] 6: NEXT
[HA1] 7: END SUB
```

Listing 27. Die Routine »verrueckt.bas«

Anwendungen gesucht

Haben Sie eine schnelle und komfortable Textverarbeitung geschrieben? Oder eine Datenbank mit frei wählbaren Eingabemasken? Eventuell ein Zeichenprogramm? Wenn Ihr Programm mit professionellen Programmen konkurrieren kann, ist es mit Sicherheit zu umfangreich, um als Listing in der DOS International zu erscheinen. Zögern Sie deshalb nicht, schicken Sie uns unverbindlich die aktuelle Version Ihres Programms und einen Ausdruck der Dokumentation. Eventuell wird Ihr Programm der Verkaufsschlager im Rahmen der neuen DOS-Software-Schiene. Falls Sie sich vorweg informieren wollen, zwischen 10.00 und 12.00 Uhr und zwischen 15.00 und 17.00 Uhr stehen wir Ihnen an allen Werktagen unter der Rufnummer (089) 4 61 00 10 zur Verfügung.

Schicken Sie Ihre Unterlagen an folgende Adresse:
 Redaktionsbüro Everts & Hagedorn
 »DOS Software«
 Hans-Stießberger-Str. 3
 8013 Haar/München

- binäre Dateien (ab Version 4.0)

Bei einer sequentiellen Datei schreiben Sie die Daten hintereinander auf den Datenträger. Das hat den Vorteil, daß man, wenn man viele Daten speichern muß, diese in einem Zug ablegen kann, denn beim Speichern auf Diskette müssen Sie das Ansteuern und Auslaufen des Diskettenlaufwerks berücksichtigen. Sind die Daten dann im Arbeitsspeicher, kann man sehr schnell auf sie zugreifen.

Sie können sequentielle, wahlfreie und binäre Dateien mit Quick Basic anlegen

Der Nachteil bei einer sequentiellen Datei liegt darin, daß Sie alle Daten bis zum Dateiende laden müssen, auch wenn Sie nur ein Datum benötigen. Auch reicht bei großen Datenmengen

eventuell der Arbeitsspeicher nicht aus, um alle Daten laden zu können. Die Datei muß dann in mehrere Abschnitte zerlegt werden.

Dateien mit wahlfreiem Zugriff sind etwas umständlicher zu programmieren. Sie bieten dafür aber wesentliche Vorteile:

- Die Daten benötigen wenig Platz auf dem Datenträger, da sie in einem gepackten binären Format gespeichert sind.
- Sie greifen auf die Dateien wahlfrei zu, das heißt, die Daten stehen irgendwo auf dem Datenträger. Der Rechner findet sie über eine Adresse (Index).

Binäre Dateien werden von Quick Basic ab Version 4.0 verwaltet. Die Dateien zeichnen sich durch eine völlig neue Struktur aus. Man kann die Lese- und Abspeicherungsform mit einem Programm vergleichen, das in

```
[MS5] 1: SUB Hscore.Drucken(Drucken(), nr)
[PN0] 2: CLS
[VP0] 3: hell
[6N1] 4: locate 12,1
[AX7] 5: PRINT Ausrichten("Modul noch nicht vorhand
en",0)
[251] 6: delay 2
[441] 7: normal
[HA1] 8: END SUB
```

Listing 28. Die Routine »hidruck« bekommt eine Meldung

```
[P89] 1: ON ERROR GOTO Dateianlegen ' Wenn keine Hs
core-Tabelle
[QZ3] 2: ' vorhanden -> d
[4B1] 3: ann Datei ' anlegen
```

Listing 29. Die Struktur des Befehls »on error goto« sehen Sie in »error.bas«

```
[TE6] 1: '-- Fehlerabfangeroutine, wenn keine ----
[UX5] 2: '----- Datei geöffnet werden kann -----
[E72] 3: Dateianlegen:
[PN0] 4: CLS
[051] 5: CLOSE #1
[RN0] 6: BEEP
[VP0] 7: hell
[V28] 8: LOCATE 12, 10: PRINT "Hscore-Tabelle wird
angelegt"
[N31] 9: Weiter
[441] 10: normal
[CU3] 11: OPEN "o", #1, "HSCORE.DAT"
[Z77] 12: PRINT #1, 1: PRINT #1, "Sechs Richtige im L
otto"
[D96] 13: PRINT #1, 2: PRINT #1, "Der Glückspilz"
[3W4] 14: PRINT #1, 3: PRINT #1, "Der Champ"
[RB6] 15: PRINT #1, 15: PRINT #1, "Der Trautmäner"
[D36] 16: PRINT #1, 20: PRINT #1, "Der Blödkoffer"
[051] 17: CLOSE #1
[X31] 18: RESUME
```

Listing 30. Die Fehleroutine finden Sie in »abfrag.bas«

Name	Datentyp	Gültigkeit	Wert in Bytes
Integer	Ganzzahl	± 32767	2
Long	Langzahl	± 2.1 ⁹	4
Real	Realzahl	± 1 ³⁸	4
Double	Doppelrealzahl	± 1 ³⁰⁸	8
String	Zeichenkette	-	4

Tabelle 2. Die verschiedenen Datentypen

Maschinensprache geschrieben ist.

Bei Superhirn speichern Sie nur wenige Daten, so daß es günstig ist, die Daten sequentiell abzulegen. Dazu müssen Sie allerdings eine Fehlerabfangeroutine vorgeben, um zu verhindern, daß Sie eine Datei öffnen wollen, die gar nicht existiert. Dies ist beispielsweise dann der Fall, wenn Sie die Hscore-Tabelle neu anlegen müssen (Listing 29).

Die Anweisung »on error goto« stellen Sie an den Programmstart. Kann eine Datei nicht geöffnet werden, weil sie nicht vorhanden ist, verzweigt Quick Basic in die Fehlerabfangeroutine.

Mit der Funktion »err/erl« spezifizieren Sie den Fehler (Listing 30).

Da beim Programmstart die Hscore-Tabelle automatisch geladen werden muß, wird beim Öffnen der Datei, falls diese nicht vorhanden ist, der Fehler »Datei nicht gefunden« ausgegeben. Aus diesem Grund müssen Sie in der Fehlerabfangeroutine die Datei als erstes mit »close« schließen. Danach öffnen Sie die Datei mit

open "o", #1, "hscore.dat"
»o« kennzeichnet den Modus »output«, um Daten zu speichern, »i« kennzeichnet den Modus »input«, um Daten zu laden, »a« kennzeichnet den Modus »append«, um die Datei zu erweitern, und »# 1« gibt die Dateinummer an, die geöffnet ist. Die Dateinummer ist ein ganzzahliger Wert im Bereich von 1 bis zu einer von MS-DOS gesetzten Obergrenze (zum Beispiel der Anweisung »files = 20«). »hscore.dat« ist der Dateiname. An dieser Stelle darf natürlich auch eine Variable stehen.

»print #dateinummer« speichert die Daten. Alle geöffneten Dateien schließen Sie mit »close«, eine bestimmte Datei mit der Anweisung »close #dateinummer«. Mit »resume«, »resume 0«, »resume next« oder »resume zeilennummer [Label]« set-

zen Sie das Programm nach einer Verzweigung in eine Fehlerabfangeroutine fort. Es fällt Ihnen sicher die Anweisung »laden(i).versuche« auf. Quick Basic ist in der Lage, Records zu verwalten, wie Sie dies auch aus anderen Programmiersprachen, etwa Pascal oder C, kennen. Records sind Daten mit unterschiedlichen Datentypen, die aber zusammengehören. Die Syntax für Records lautet:

```
type Recordname
Elementenname as Datentyp
Elementenname as Datentyp
end type
```

»Recordname« ist die Kennzeichnung des Records, zum Beispiel »familiendatei«, »kartenspiel« oder »hscore-tabelle«.

Elementenname ist der Variablenname - quasi eine Untergliederung des Records. Datentyp steht für »integer«, »longinteger«, »real«, »double« oder »string * Zeichenkettenlänge«. Die Gültigkeit der einzelnen Variablentypen und der entsprechenden Speicherbelegung entnehmen Sie Tabelle 2.

Wenn Sie zum Beispiel eine Mitarbeiterdatei einer Firma als Record verwalten möchten, bietet sich folgende Struktur an:

```
Vorname: Zeichenkette (Länge 10)
Nachname: Zeichenkette (Länge 20)
Wohnort: Zeichenkette (Länge 15)
PLZ: Ganzzahl
Gehalt: Langzahl
Umsatz: Langzahl
Als Record sieht es dann folgendermaßen aus:
```

```
type firma
vorname as string * 10
nachname as string * 20
wohnort as string * 15
plz as integer
gehalt as long
umsatz as long
end type
```

Die strenge Zuordnung der Datentypen ist zwingend notwendig. Aber auch im »täglichen Hausgebrauch« sollte man sich Gedanken über den Gültigkeitsbereich der Varia-


```
[HM3] 1: CALL Upro(Datei(), max)
[000] 2:
[WZ4] 3: SUB Upro(Drucken() AS Firma, n)
[MU3] 4:   Recordbezeichnung ---^
[HV1] 5:   FOR i=1 to n
[V34] 6:     PRINT Drucken(i).Vorname
[SE3] 7:     PRINT Drucken(i).Name
[W20] 8:   .
[W20] 9:   .
[ZS0] 10:  NEXT
[HA1] 11: END SUB
```

Listing 31. Wie Sie Records übergeben, zeigt »upro.bas«

```
[524] 1: CALL Hiscore.Laden(Daten(), nr)
[000] 2:
[686] 3: SUB Hiscore.Laden(Laden() AS Hiscore, i)
[MT3] 4: OPEN "i", #1, "HISCORE.DAT"
[NB0] 5: DO
[611] 6:   i = i + 1
[4B4] 7:   INPUT #1, Laden(i).Versuche
[MM3] 8:   INPUT #1, Laden(i).Namen
[ZF3] 9: LOOP UNTIL (EOF(1) = -1)
[051] 10: CLOSE #1
[HA1] 11: END SUB
```

Listing 32. Die Hiscore-Tabelle laden Sie wie in »hiscore.bas«

```
[WD2] 1: DEFINT I-J, X-Y
[R87] 2: SUB Hiscore.Drucken (Drucken() AS Hiscore,
n)
[PN0] 3:   CLS
[VP0] 4:   Hell
[YS1] 5:   LOCATE 1, 20:
[HY5] 6:   PRINT Mitte("   Hiscore-Tabelle   ", "*")
[441] 7:   normal
[DN7] 8:   PRINT "LfdNr."; TAB(20); "Versuche"; TAB(
40); "Namen"
[W63] 9:   PRINT STRING$(80, "-")
[K32] 10:  FOR i = 1 TO n
[FB1] 11:    PRINT i;
[BN5] 12:    PRINT TAB(20); Drucken(i).Versuche;
[HX4] 13:    PRINT TAB(40); Drucken(i).Namen
[ZS0] 14:  NEXT
[N31] 15:  Weiter
[HA1] 16: END SUB
```

Listing 33. Den aktuellen Hiscore-Stand erhalten Sie durch »hisdruck.bas«

```
[GX4] 1: ***** Hiscore-Tabelle *****
[000] 2:
[N94] 3: FUNCTION Mitte$(Text$, x$)
[GE4] 4:   rest = INT((80 - LEN(Text$)) / 2)
[MZ7] 5:   Mitte$ = STRING$(rest, x$) + Text$ + STRIN
G$(rest, x$)
[R52] 6: END FUNCTION
```

Listing 34. Mit »mitte.bas« grenzen Sie einen Text mit definierten Zeichen ein

```
[044] 1: ' -----
[485] 2: '               Konstantendeklaration
[466] 3: '
[US2] 4:   RANDOMIZE TIMER
[N77] 5:   CONST Lauftext$ = "* C by DOS - Internati
onal "
[C53] 6:   CONST Version = 2.17
[EL2] 7:   CONST Ser.Nr = 88
[702] 8:   CONST max = 4
[702] 9:   CONST maxHi = 20
[6L2] 10:  CONST TabMax = 10
[TT8] 11:  DIM Spieler(max) AS INTEGER, Rechner(max)
AS INTEGER
[000] 12:
[762] 13:  TYPE Hiscore
[LG3] 14:    Versuche AS INTEGER
```

```
[F33] 15:  Namen AS STRING * 25
[WE1] 16:  END TYPE
[6S4] 17:  DIM Daten(maxHi) AS Hiscore
[000] 18:
[W16] 19:  ' -----

[000] 20:
[P89] 21: ON ERROR GOTO Dateianlegen ' Wenn keine His
core-Tabelle
[023] 22:   ' vorhanden -> d
ann Datei
[4B1] 23:   ' anlegen
[A42] 24: 'KILL "hisc*.*"
[ZV3] 25: '----- Programmtest
[000] 26:
[000] 27:
[PN0] 28: CLS
[C31] 29: blinkt
[Y02] 30: GOSUB Cover
[591] 31: heulen 2
[W14] 32: delay 1 'wartet 1 Sekunde
[PN0] 33: CLS
[441] 34: normal
[Y02] 35: GOSUB Cover
[T86] 36: LOCATE 4, 5: PRINT "Version: "; Version;
[GP5] 37: LOCATE 5, 5: PRINT "Ser-Nr. "; Ser.Nr;
[AU4] 38: CALL LaufSchrift(Lauftext$, 2)
[D93] 39: ' 2 x laufen -----^
[000] 40:
[524] 41: CALL Hiscore.Laden(Daten(), nr)
[795] 42: CALL Hiscore.Drucken(Daten(), nr)
[BV0] 43: hilfe
[RM5] 44: CALL NamenAufnehmen(Spielername$)
[000] 45:
[524] 46: CALL Hiscore.Laden(Daten(), nr)
[795] 47: CALL Hiscore.Drucken(Daten(), nr)
[BV0] 48: hilfe
[RM5] 49: CALL NamenAufnehmen(Spielername$)
[000] 50:
[NB0] 51: DO
[0W0] 52: Menue
[A42] 53: CALL Taste(x$)
[U72] 54: SELECT CASE x$
[631] 55: CASE "1"
[RM5] 56:   CALL NamenAufnehmen(Spielername$)
[000] 57:
[A31] 58: CASE "2"
[2U4] 59:   CALL Hiscore.Sort(Daten(), nr)
[HX4] 60:   IF nr > TabMax THEN nr = TabMax
[795] 61:   CALL Hiscore.Drucken(Daten(), nr)
[000] 62:
[E31] 63: CASE "3"
[VK1] 64:   nr = nr + 1
[HW6] 65:   IF nr = hiMax THEN nr = TabMax + 1 ' nr
> maxHi
[TN7] 66:   CALL Spielen(Rechner(), Spieler(), max,
Versuche)
[WF6] 67:   IF Versuche < Daten(nr - 1).Versuche TH
EN
[NX4] 68:     Daten(nr).Namen = Spielername$
[KV4] 69:     Daten(nr).Versuche = Versuche
[M93] 70:     Hiscore.Aendern = -1
[801] 71:   END IF
[000] 72:
[K31] 73: CASE "4"
[BV0] 74:   hilfe
[000] 75:
[P31] 76: CASE "5"
[Z84] 77:   IF Hiscore.Aendern = -1 THEN
[2U4] 78:     CALL Hiscore.Sort(Daten(), nr)
[HX4] 79:     IF nr > TabMax THEN nr = TabMax
[GY7] 80:     IF nr <> 0 THEN CALL Hiscore.Speiche
rn(Daten(), nr)
[801] 81:   END IF
[9P0] 82:   Ende
[Y02] 83:   GOSUB Cover
[591] 84:   heulen 2
[Y68] 85:   CALL LaufSchrift("* ENDE * ENDE * END
E * ENDE * ENDE ", 2)
[Y41] 86:   delay 1
[PN0] 87:   CLS
[4M0] 88:   END
[000] 89:
[PF1] 90: CASE ELSE
[721] 91:   Fehler
[GS1] 92: END SELECT
[HS0] 93: LOOP
```

Listing 35. Die neue Fassung des Hauptprogramms »haupt3.bas«

blen machen, denn gerade Ganzzahlen verarbeitet Basic wesentlich schneller als Realzahlen. Daher ist es sinnvoll, Laufvariablen immer als Integer zu definieren. Werden Zahlen erwartet, die außerhalb des Gültigkeitsbereichs liegen, müssen Sie prüfen, ob der Datentyp »long« ausreicht.

Die Zuordnung erfolgt mit der Dim-Anweisung

```
dim datei as firma
```

und die Verkettung des Oberbegriffs und der einzelnen Elemente mit Hilfe des Punkts, beispielsweise

```
datei.vorname = "Heinz"
```

```
datei.nachname = "Müller"
```

```
:
```

```
:
```

```
datei.umsatz = 210000
```

Wollen Sie den Record als Bereich definieren, legen Sie mit der Anweisung »dim« die Tiefe fest.

```
dim datei(max) as firma
```

Die Daten weisen Sie jetzt, wie bei den Bereichen üblich, über einen Index zu.

```
datei(i).vorname = "Heinz"
datei(i).nachname = "Müller"
```

```
:
```

```
:
```

```
datei(i).umsatz = 210000
```

Besonders wichtig ist, daß man einen Record in seiner Gesamtheit an eine Prozedur übergibt. Es ist erforderlich, daß Sie in der Prozedur den Record mittels Recordbezeichnung festlegen. Hätten Sie keinen Record verwendet, müßten Sie die Daten recht umständlich und »tippintensiv« übergeben, zum Beispiel mit:

```
call upro(namen(), vorname(),
wohnort(), plz(), gehalt(),
umsatz(), max)
```

Wie ein Record übergeben wird, sehen Sie in Listing 31. Beim Superhirn-Programm wäre der Einsatz eines Re-

```
[ØV6] 1: SUB Hscore.Sort (Sort() AS Hscore, nr)
[ØV6] 2: FOR i = 1 TO nr - 1
[ØV6] 3: FOR j = i + 1 TO nr
[ØV6] 4: IF Sort(i).Versuche > Sort(j).Versuche THEN
[ØV6] 5: SWAP Sort(i).Versuche, Sort(j).Versuche
[ØV6] 6: SWAP Sort(i).Namen, Sort(j).Namen
[ØV6] 7: END IF
[ØV6] 8: NEXT j, i
[ØV6] 9: END SUB
```

Listing 36. Die Sortierroutine »sort.bas«

```
[F88] 1: SUB Hscore.Speichern (Speichern() AS Hscore, max)
[CU3] 2: OPEN "o", #1, "HSCORE.DAT"
[BE2] 3: FOR i = 1 TO max
[835] 4: PRINT #1, Speichern(i).Versuche
[SC4] 5: PRINT #1, Speichern(i).Namen
[ZSØ] 6: NEXT
[HA1] 7: END SUB
```

Listing 37. »speicher.bas« überträgt die Informationen der Hscore-Tabelle auf Diskette

```
[DS3] 1: SUB Zahl.Aufnehmen(...)
[W2Ø] 2: .
[W2Ø] 3: .
[K46] 4: ELSEIF ASC(x$) >= 48 AND ASC(x$) <= 57 THEN
N
[2L4] 5: ' 1 bis Ø -----^-----^
[611] 6: i = i + 1
[4L2] 7: IF i = max + 1 THEN
[YP2] 8: PRINT "<ENTER>";
[TE1] 9: EXIT SUB
[ØØ1] 10: END IF
[323] 11: Spieler(i) = VAL(x$)
[A17] 12: CALL Pruefen(Spieler(), Spieler(i), i, Test)
[SP2] 13: IF Test < -1 THEN
[145] 14: LOCATE y, x + 1: PRINT Spieler(i);
[ØØ1] 15: END IF
```

Listing 38. Das Prüfmodul »pruef2.bas« mit erweiterten Funktionen

Für Superhirn-Spieler:

Wer den Grundlagenartikel nicht durcharbeiten, sondern nur das Programm »Superhirn« besitzen will, der benötigt allein die Listings 6 (»super.cov«), 41 (»super4.bas«) und 42 (»prog-um.bas«), um Superhirn zu einem Spielchen zu bewegen.

cords nicht zwingend notwendig, da Sie nur zwei Werte speichern wollen, nämlich den Spielernamen und die Anzahl der Versuche.

```
type hscore
versuche as integer
namen as string * 25
end type
dim daten(maxhi) as hscore
```

Wie die Hscore-Tabelle geladen wird, zeigt Listing 32.

Die Datei öffnen Sie mit »open "i"«. Sie nehmen so lange die Daten auf, bis das Zeichen »eof« (Dateiende) auf »-1« gesetzt ist. Sie speichern die Daten entsprechend der Bestimmung in »versuche« und »namen« ab. Sobald die Hscore-Tabelle geladen ist, drucken Sie sie aus, damit der Spieler gleich weiß, »was die Stunde geschlagen hat« (Listing 33).

Was bedeutet in diesem Zusammenhang die Anweisung »print mitte(" Text ", "-")«? Ähnlich dem Problem, Texte zentriert auszurichten (Funktion »ausrichten«), ist es oft der Fall, daß man einen Text durch bestimmte Zeichen begrenzen will, meist durch »*«, »#«, »-«, »:«, »=«, »\$« und so weiter. Sie lösen das Problem wieder über eine Funktion. Sehen Sie sich als Beispiel dazu das Listing 34 an. Die Funktion »mitte\$« wäre wiederum ein passendes Tool für die Programmumgebung.

Nun müssen Sie noch folgendes Problem lösen: Wann nehmen Sie jemand in die Hscore-Tabelle auf? Natürlich nur, wenn er weniger Versuche hat als der schlechteste Spieler, der auf der »maxhi.Stelle« ist. Hier sollten Sie wiederum mit einer

```
[ZD2] 1: DEFINT E, I, N, T
[AL6] 2: SUB Pruefen (Test(), Eingabe, n, Doppelt)
[XV1] 3: Doppelt = Ø
[KF2] 4: FOR i = 1 TO n - 1
[W86] 5: IF Test(i) = Eingabe THEN 'Zahl doppelt
[NN1] 6: y = CSRLIN
[ØG1] 7: x = POS(Ø)
[2X1] 8: Doppelt = -1
[221] 9: n = n - 1
[RNØ] 10: BEEP
[SM1] 11: LOCATE 24, 1
[ØH1] 12: intensiv
[E97] 13: PRINT Mitte(" Zahl doppelt eingegeben "
, "-");
[441] 14: normal
[251] 15: delay 2
[7N1] 16: Fusszeile
[W36] 17: LOCATE y, x 'Cursor zurückstellen
[AD1] 18: EXIT FOR
[ØØ1] 19: END IF
[ZSØ] 20: NEXT
[HA1] 21: END SUB
```

Listing 39. Das eigentliche Modul »pruef3.bas«

```
[TL1] 1: DEFINT A-Z
[PC1] 2: SUB Ende
[C31] 3: blinkt
[6MØ] 4: x = 1
[V73] 5: FOR y = 1 TO 11 STEP 2
[A94] 6: CALL Rahmen(x, y, 8Ø - x, 24 - y)
[631] 7: x = x + 3
[ZSØ] 8: NEXT
[441] 9: normal
[ØØØ] 10:
[ZA6] 11: LOCATE 12, 26: PRINT "***** E N D E *****"
[251] 12: delay 2
[PNØ] 13: CLS
[HA1] 14: END SUB
```

Listing 40. Das Abschlußmodul »ende.bas«

Konstanten arbeiten, denn der eine Benutzer geht davon aus, daß zehn Spieler in der Tabelle ausreichen, der zweite möchte die Namen der besten zwanzig Spieler ausgedruckt haben.

In der »Ulk-Hiscore-Tabelle« aus Listing 30 haben schon fünf Spieler gespielt. Die Ergebnisse liegen zwischen 1 und 20. Der nächste Spieler geht mit der Nummer sechs an den Start. Nach der Auswertung des Spiels wird in das Hauptprogramm zurückgeschaltet. Hier muß die Abfrage erfolgen: »War ich gut genug, um in die Hiscore-Tabelle aufgenommen zu werden?«. Außerdem müssen Sie vermerken, daß man die Hiscore-Tabelle am Ende ändern und speichern muß (»hiscore.aendern = -1«). Das Hauptprogramm präsentiert sich nun entsprechend. Beachten Sie besonders die Anweisung nach »case "3"« in Listing 35.

Natürlich müssen Sie die Hiscore-Tabelle auch sortieren. Sie wählen dafür einen

einfachen Sortieralgorithmus »Sortieren durch Austauschen« (Listing 36). Am Ende des Programms speichern Sie die Hiscore-Tabelle sequentiell. Die Daten legen Sie mit »print #dateinummer«

auf dem Datenträger ab (Listing 37).

Einige Programmtester werden bemängeln, daß das Programm nicht überprüft, ob Zahlen doppelt eingegeben wurden. Als Entwickler kommt man kaum darauf, daß jemand die Zahlen »1 1 1« eingibt, da ja die Forderung gestellt ist, mit so wenigen Abfragen wie möglich die Superhirn-Zahl zu erraten. Deshalb ist es sehr wichtig, Programme von verschiedenen Benutzern testen zu lassen, um ein »Feedback« zu erhalten und weitere Plausibilitätskontrollen einzubauen.

Die Überprüfungsprozedur ist schnell geschrieben. Nachdem eine Zahl vom Spieler eingegeben wurde, muß überprüft werden, ob diese Zahl schon einmal eingegeben wurde (Listing 38).

Die Prozedur »prüfen« (Listing 39) überprüft von »i = 1« bis »n-1«, ob eine Zahl doppelt vorhanden ist. Bei einer Fehlerbedingung wird »doppelt« auf »-1« gesetzt. Beachten Sie, daß Sie die Cursor-Koordinaten wiederum zwischenspeichern müssen. Am Ende des Programms (Listing 40) muß sich das Spiel natürlich mit einem richtigen »Spektakel« verabschieden. Dazu sollten Sie wieder alle Hilfsmittel einsetzen (siehe Hauptprogramm).

Lassen Sie Ihre Programme von anderen testen – Sie werden staunen, was dabei alles zutage kommt

Das Programm Superhirn ist inzwischen auf über 14 KByte angewachsen – und dabei doch überschaubar geblieben. Das liegt in erster Linie an der modularen Bauweise. Sie haben insgesamt 31 Prozeduren und drei Funktionen programmiert.

Jetzt könnten Sie noch weitere Raffinessen einbauen oder das Spiel soweit modifizieren, daß die Spielstärke vom Spieler festgesetzt werden kann (Anzahl der zu ratenden Ziffern). Weiter können Sie vereinbaren, daß zum Beispiel von Spielstärke sechs (sechs Ziffern) an auch Zahlen doppelt vorkommen dürfen. Der Phantasie sind keine Grenzen gesetzt.

Der Zweck dieses kleinen Spiels liegt darin, zu beweisen, daß man in Quick Basic sehr wohl vernünftig und strukturiert programmieren kann. Das Hauptprogramm umfaßt nur eine Seite und ließe sich sogar noch kürzen. Das Programm verwendet nur ein Label, um das Unterprogramm »cover« aufzurufen. Wie sinnvoll strukturierte Programmierung ist, sieht man unter anderem auch daran, daß ein Pascal- oder C-Programmierer Superhirn in kurzer Zeit in seine Sprache übersetzt hat.

(Gerd Kebschull/gh)

```
[WA7] 1: DECLARE FUNCTION Ausrichten$ (Text$, minus!
[GM5] 2: DECLARE FUNCTION Mitte$ (Text$, x$)
[8F4] 3: DECLARE FUNCTION Datum$ (d$)
[000] 4:
[3A8] 5: DECLARE SUB Pruefen (Test$(), Eingabe$, n$,
        Doppelt!)
[2X6] 6: DECLARE SUB Loesung (Richtig$(), Schluss$)
[KR3] 7: DECLARE SUB heulen (n$)
[GH5] 8: DECLARE SUB DuBistVerrueckt (n!)
[RW8] 9: DECLARE SUB Hiscore.Speichern (Speichern()
        AS ANY, max$)
[EB7] 10: DECLARE SUB Hiscore.Laden (Laden() AS ANY,
        max$)
[RH7] 11: DECLARE SUB Hiscore.Sort (Sort() AS ANY, nr
        $)
[TZ7] 12: DECLARE SUB Hiscore.Drucken (Drucken() AS A
        NY, n$)
[9B3] 13: DECLARE SUB Fehler ()
[S23] 14: DECLARE SUB Fusszeile ()
[3D4] 15: DECLARE SUB Zahl.holen (x$)
[D23] 16: DECLARE SUB Ende ()
[XLA] 17: DECLARE SUB Ueberpruefen (Rechner$(), Spiel
        er$(), n$, Zahl!, Platz!)
[LZ8] 18: DECLARE SUB Zahl.Aufnehmen (Spieler() AS IN
        TEGER, max$)
[AK9] 19: DECLARE SUB Spielen (Rechner$(), Spieler$()
        , max$, Versuche!)
[UN8] 20: DECLARE SUB Zahlen.Erzeugen (Super() AS INT
        EGER, n$)
[6N5] 21: DECLARE SUB Auswertung (Versuch!)
[WY6] 22: DECLARE SUB NamenAufnehmen (Spielernamen$)
[NM3] 23: DECLARE SUB Mainzel ()
[615] 24: DECLARE SUB Ausgabe (Versuch!)
[4N6] 25: DECLARE SUB LaufSchrift (Text$, Anzahl!)
[RN5] 26: DECLARE SUB Rahmen (x1$, y1$, x2$, y2$)
[2W5] 27: DECLARE SUB Loesche (x1$, y1$, x2$, y2$)
[414] 28: DECLARE SUB delay (Zeit!)
[8N6] 29: DECLARE SUB klein.Rahmen (x1$, y1$, x2$, y2
        $)
[EC3] 30: DECLARE SUB blinkt ()
[Z23] 31: DECLARE SUB hell ()
[6D3] 32: DECLARE SUB normal ()
[4X3] 33: DECLARE SUB intensiv ()
[RC3] 34: DECLARE SUB weiter ()
[SE3] 35: DECLARE SUB taste (x$)
[H63] 36: DECLARE SUB Menue ()
```

```
[W53] 37: DECLARE SUB hilfe ()
[000] 38:
[NKT] 39:
[7M6] 40: ' | Datei : SUPER4.BAS
[AY5] 41: ' | Stand : 08.12.88
[5Z5] 42: ' | Update : 23.12.88
[K07] 43: ' | Autor : Gerd Kebschull
[Z18] 44: ' | Software: Microsoft QuickBASIC 4.0
[035] 45: '
[S67] 46: ' | C by DOS-International
[NKT] 47:
[000] 48:
[8W5] 49:
[485] 50: ' Konstantendeklaration
[466] 51: '
[US2] 52: RANDOMIZE TIMER
[N77] 53: CONST Lauftext$ = " C by DOS - Internati
        onal "
[C53] 54: CONST Version = 2.23
[EL2] 55: CONST Ser.Nr = 88
[T12] 56: 'CONST max = 6
[TG2] 57: CONST maxHi = 20
[6L2] 58: CONST TabMax = 10
[2B5] 59: DIM max AS INTEGER, nr AS INTEGER
[UD8] 60: 'DIM Spieler(max) AS INTEGER, Rechner(max
        ) AS INTEGER
[000] 61:
[000] 62:
[762] 63: TYPE Hiscore
[LG3] 64: Versuche AS INTEGER
[F33] 65: Namen AS STRING * 25
[WEL] 66: END TYPE
[6S4] 67: DIM Daten(maxHi) AS Hiscore
[000] 68:
[W16] 69:
[000] 70:
```



```

[P89] 71: ON ERROR GOTO Dateianlegen ' Wenn keine His
core-Tabelle
[Ø23] 72: ' vorhanden -> d
ann Datei
[4B1] 73: ' anlegen
[A42] 74: 'KILL "hisc*.*"
[ZV3] 75: ' ^----- Programmtest
[ØØØ] 76:
[ØØØ] 77:
[PNØ] 78: CLS
[C31] 79: blinkt
[YØ2] 80: GOSUB Cover
[591] 81: heulen 2
[694] 82: delay 1 'wartet 1 Sekunden
[PNØ] 83: CLS
[441] 84: normal
[YØ2] 85: GOSUB Cover
[T86] 86: LOCATE 4, 5: PRINT "Version: "; Version;
[GP5] 87: LOCATE 5, 5: PRINT "Ser-Nr. "; Ser.Nr;
[AU4] 88: CALL LaufSchrift(LaufText$, 2)
[D93] 89: ' 2 x laufen ^-----
[ØØØ] 90:
[524] 91: CALL Hiscore.Laden(Daten(), nr)
[795] 92: CALL Hiscore.Drucken(Daten(), nr)
[BVØ] 93: hilfe
[ØØØ] 94:
[RM5] 95: CALL NamenAufnehmen(Spielername$)
[NBØ] 96: DO
[RNØ] 97: BEEP
[PNØ] 98: CLS
[Z97] 99: LOCATE 12, 1Ø: PRINT "Spielstärke eingeben
2-6"
[A42] 100: CALL taste(x$)
[P12] 101: max = VAL(x$)
[VX4] 102: LOOP UNTIL (max <= 6 AND max > 1)
[ØØØ] 103:
[TT8] 104: DIM Spieler(max) AS INTEGER, Rechner(max) A
S INTEGER
[ØØØ] 105:
[ØØØ] 106:
[NBØ] 107: DO
[ØWØ] 108: Menue
[A42] 109: CALL taste(x$)
[U72] 110: SELECT CASE x$
[631] 111: CASE "1"
[RM5] 112: CALL NamenAufnehmen(Spielername$)
[F83] 113: Hiscore.Aendern = Ø
[ØØØ] 114:
[A31] 115: CASE "2"
[2U4] 116: CALL Hiscore.Sort(Daten(), nr)
[HX4] 117: IF nr > TabMax THEN nr = TabMax
[795] 118: CALL Hiscore.Drucken(Daten(), nr)
[ØØØ] 119:
[E31] 120: CASE "3"
[VK1] 121: nr = nr + 1
[M24] 122: IF nr = hiMax THEN nr = TabMax + 1
[2W7] 123: CALL Spielen(Rechner(), Spieler(), max,
Versuche)
[WF6] 124: IF Versuche < Daten(nr - 1).Versuche TH
EN
[NX4] 125: Daten(nr).Namen = Spielername$
[KV4] 126: Daten(nr).Versuche = Versuche
[M93] 127: Hiscore.Aendern = -1
[5F4] 128: 'PRINT "Hiscore geändert"
[FB1] 129: 'weiter
[8Ø1] 130: END IF
[K31] 131: CASE "4"
[BVØ] 132: hilfe
[P31] 133: CASE "5"
[Z84] 134: IF Hiscore.Aendern = -1 THEN
[2U4] 135: CALL Hiscore.Sort(Daten(), nr)
[HX4] 136: IF nr > TabMax THEN nr = TabMax
[GY7] 137: IF nr <> Ø THEN CALL Hiscore.Speich
ern(Daten(), nr) ^--- wurde gespielt ?
[L54] 138:
[8Ø1] 139: END IF
[9PØ] 140: Ende
[YØ2] 141: GOSUB Cover
[591] 142: heulen 2
[Y68] 143: CALL LaufSchrift(" ENDE * ENDE * EN
DE * ENDE * ENDE ", 2)
[Y41] 144: delay 1
[PNØ] 145: CLS
[4MØ] 146: END
[ØØØ] 147:
[PF1] 148: CASE ELSE
[721] 149: Fehler
[GS1] 150: END SELECT
[HSØ] 151: LOOP
[ØØØ] 152:
[4MØ] 153: END
[ØØØ] 154:

```

```

[FM6] 155: '----- Fehlerabfangroutine, wenn keine ----
[6P6] 156: '----- Datei geöffnet werden kann -----
[E72] 157: Dateianlegen:
[PNØ] 158: CLS
[Ø51] 159: CLOSE #1
[RNØ] 160: BEEP
[VPØ] 161: hell
[V28] 162: LOCATE 12, 1Ø: PRINT "Hiscore-Tabelle wird
angelegt"
[N31] 163: weiter
[441] 164: normal
[CU3] 165: OPEN "o", #1, "HISCORE.DAT"
[Z77] 166: PRINT #1, 1: PRINT #1, "Sechs Richtige im
Lotto"
[D96] 167: PRINT #1, 2: PRINT #1, "Der Glückspilz"
[3W4] 168: PRINT #1, 3: PRINT #1, "Der Champ"
[RB6] 169: PRINT #1, 15: PRINT #1, "Der Traumtänzer"
[D36] 170: PRINT #1, 2Ø: PRINT #1, "Der Blödkoffer"
[Ø51] 171: CLOSE #1
[X31] 172: RESUME
[ØØØ] 173:
[ØØØ] 174:
[4MØ] 175: END
[TØ4] 176: REM $INCLUDE: 'super.cov'
[ØØØ] 177:
[HK1] 178: DEFINT I-J
[ØL3] 179: SUB Ausgabe (Versuch)
[AW3] 180: CALL Loesche(19, 11, 6Ø, 13)
[ØH1] 181: intensiv
[LH4] 182: CALL klein.Rahmen(19, 11, 6Ø, 13)
[441] 183: normal
[HR9] 184: LOCATE 12, 24: PRINT " Sie haben "; Versuc
h; " Versuche benötigt"
[251] 185: delay 2
[HA1] 186: END SUB
[ØØØ] 187:
[MH1] 188: DEFSG I-J
[SZ5] 189: FUNCTION Ausrichten$(Text$, minus)
[CN5] 190: rest = INT((8Ø - LEN(Text$)) / 2) - minus
[HY3] 191: 'zieht minus-Zeichen ab
[2V8] 192: Ausrichten$ = STRING$(rest, " ") + Text$ +
STRING$(rest, " ")
[R52] 193: END FUNCTION
[ØØØ] 194:
[HK1] 195: DEFINT I-J
[184] 196: SUB Auswertung (Versuch)
[PNØ] 197: CLS
[8Y2] 198: IF Versuch < 8 THEN
[C31] 199: blinkt
[X53] 200: CALL Rahmen(19, 11, 6Ø, 13)
[SK7] 201: LOCATE 12, 22: PRINT "**** S U P E R L E
I S T U N G ****"
[2C1] 202: Mainzel
[441] 203: normal
[GG3] 204: CALL Ausgabe(Versuch)
[8Ø1] 205: END IF
[ØØØ] 206:
[A36] 207: IF (Versuch >= 8) AND (Versuch <= 11) THEN
[G37] 208: Text$ = "* G U T E Leistung* G U T E L
eistung"
[192] 209: FOR y = 8 TO 16
[KX4] 210: LOCATE y, 1: PRINT Text$; Text$
[ZSØ] 211: NEXT
[Z34] 212: CALL LaufSchrift(Text$, 1)
[GG3] 213: CALL Ausgabe(Versuch)
[8Ø1] 214: END IF
[ØØØ] 215:
[L26] 216: IF (Versuch > 11) AND (Versuch <= 14) THEN
[ZV7] 217: LOCATE 12, 2Ø: PRINT "**** befriedigende
LEISTUNG ****"
[251] 218: delay 2
[GG3] 219: CALL Ausgabe(Versuch)
[8Ø1] 220: END IF
[ØØØ] 221:
[ZV5] 222: IF (Versuch > 14) AND (Versuch < 18) THEN
[C31] 223: blinkt
[MR8] 224: LOCATE 12, 2Ø: PRINT "**** Das war nicht
besonders stark ****"
[543] 225: DuBistVerrueckt 2
[Y41] 226: delay 1
[GG3] 227: CALL Ausgabe(Versuch)
[8Ø1] 228: END IF
[FG3] 229: IF (Versuch >= 18) THEN
[PNØ] 230: CLS
[CD9] 231: LOCATE 11, 2Ø: PRINT "Wir brechen hier
an dieser Stelle ab."
[B7A] 232: LOCATE 12, 2Ø: PRINT "Bitte nicht RATEN s
ondern logisch denken!"

```

Listing 41. »super4.bas« ist die endgültige Version von Superhirn


```

[543] 233: DuBistVerrueckt 2
[Y41] 234: delay 1
[GG3] 235: CALL Ausgabe(Versuch)
[N31] 236: weiter
[801] 237: END IF
[HA1] 238: END SUB
[000] 239:
[MH1] 240: DEF SNG I-J
[D43] 241: FUNCTION Datum$(d$)
[C18] 242: Datum$ = MID$(d$, 4, 2) + "." + LEFT$(d$,
2) + "." + RIGHT$(d$, 4)
[R52] 243: END FUNCTION
[000] 244:
[C64] 245: SUB DuBistVerrueckt (n)
[K32] 246: FOR i = 1 TO n
[425] 247: PLAY "o2 f4 p6 a+ p6 d8 f f f p4
"
[VV5] 248: Du bist verrueckt mein Kind
[585] 249: a$ = INKEY$: IF a$ <> "" THEN EXIT SUB
[ZS0] 250: NEXT
[HA1] 251: END SUB
[000] 252:
[TL1] 253: DEFINT A-Z
[PC1] 254: SUB Ende
[C31] 255: blinkt
[6M0] 256: x = 1
[V73] 257: FOR y = 1 TO 11 STEP 2
[A94] 258: CALL Rahmen(x, y, 80 - x, 24 - y)
[631] 259: x = x + 3
[ZS0] 260: NEXT
[441] 261: normal
[000] 262:
[ZA6] 263: LOCATE 12, 26: PRINT "***** E N D E
*****"
[251] 264: delay 2
[PN0] 265: CLS
[HA1] 266: END SUB
[000] 267:
[WK1] 268: DEF SNG A-Z
[MS1] 269: SUB Fehler
[VP0] 270: hell
[491] 271: heulen 1
[SM1] 272: LOCATE 24, 1
[6E6] 273: PRINT Mitte$(" " >>> falsche Angabe <<< " , "
-");
[441] 274: normal
[Y41] 275: delay 1
[HA1] 276: END SUB
[000] 277:
[GZ1] 278: DEFINT M, R-S
[MB2] 279: SUB Fusszeile
[VP0] 280: hell
[3WA] 281: LOCATE 24, 1: PRINT "Zahl löschen mit '#
- bestätigen mit <Enter>";
[RB3] 282: PRINT STRING$(32, " ");
[441] 283: normal
[HA1] 284: END SUB
[000] 285:
[H32] 286: DEFINT I, N, T
[LY1] 287: DEF SNG M, R-S
[Z72] 288: SUB heulen (n)
[K32] 289: FOR i = 1 TO n
[6A4] 290: FOR ton = 40 TO 2000 STEP 100
[EL1] 291: a$ = INKEY$
[GG3] 292: IF a$ <> "" THEN EXIT SUB
[VF5] 293: -- Abbruch nach Tastendruck
[MZ1] 294: SOUND ton, .5
[ZS0] 295: NEXT
[ZS0] 296: NEXT
[HA1] 297: END SUB
[000] 298:
[M22] 299: DEF SNG I, N, T
[SH1] 300: SUB hilfe
[PN0] 301: CLS
[093] 302: CALL Rahmen(1, 1, 80, 4)
[VP0] 303: hell
[P76] 304: LOCATE 2, 25: PRINT " S U P E R H I R
N
[XY7] 305: LOCATE 3, 25: PRINT "
[WC2] 306: PRINT : normal
[R29] 307: PRINT "Der Rechner hat eine vierstellige
Zufallszahl";
[086] 308: PRINT " erzeugt. Es wird keine Ziffer"
[FT8] 309: PRINT "doppelt vorkommen. Der Spieler gibt
ebenfalls";
[BB6] 310: PRINT " eine vierstellige Zahl ein und"
[XH6] 311: PRINT "bekommt folgende Informationen:"
[VP0] 312: hell
[CT4] 313: PRINT : PRINT " 2.3
[441] 314: normal

```

```

[BP8] 315: PRINT "Dies bedeutet, er hat insgesamt dre
i Ziffern";
[SP6] 316: PRINT " richtig, wovon zwei auch schon"
[NF6] 317: PRINT "auf dem richtigen Platz sind (1,4).
"
[AR2] 318: PRINT "Beispiel:"
[0H1] 319: intensiv
[XN4] 320: PRINT : PRINT " -> 1 3 6 4 <-";
[3A3] 321: PRINT " Super - Zahl "
[441] 322: normal
[FA5] 323: PRINT " 1 2 3 4 <- Spieler-Zahl"
[VP0] 324: hell
[EU3] 325: LOCATE 18, 10: PRINT " 2.3 "
[441] 326: normal
[PH4] 327: CALL klein.Rahmen(1, 19, 80, 22)
[80B] 328: LOCATE 20, 2: PRINT Ausrichten("Lauftexte
und Töne können durch", 1)
[XHC] 329: LOCATE 21, 2: PRINT Ausrichten("Drücken ei
ner Taste abgebrochen werden", 1)
[N31] 330: weiter
[HA1] 331: END SUB
[000] 332:
[YL2] 333: DEFINT I-J, N, X-Y
[R87] 334: SUB Hiscore.Drucken (Drucken() AS Hiscore,
n)
[PN0] 335: CLS
[VP0] 336: hell
[YS1] 337: LOCATE 1, 20:
[HY5] 338: PRINT Mitte(" Hiscore-Tabelle ", "")
[441] 339: normal
[DN7] 340: PRINT "LfdNr."; TAB(20); "Versuche"; TAB(
40); "Namen"
[W63] 341: PRINT STRING$(80, "-")
[K32] 342: FOR i = 1 TO n
[BF3] 343: PRINT seite - 1 * 20 + i;
[FB1] 344: PRINT i;
[BN5] 345: PRINT TAB(20); Drucken(i).Versuche;
[HX4] 346: PRINT TAB(40); Drucken(i).Namen
[ZS0] 347: NEXT
[N31] 348: weiter
[HA1] 349: END SUB
[000] 350:
[EB2] 351: DEF SNG J, N, X-Y
[686] 352: SUB Hiscore.Laden (Laden() AS Hiscore, i)
[MT3] 353: OPEN "i", #1, "HISCORE.DAT"
[NB0] 354: DO
[611] 355: i = i + 1
[4B4] 356: INPUT #1, Laden(i).Versuche
[MM3] 357: INPUT #1, Laden(i).Namen
[ZF3] 358: LOOP UNTIL (EOF(1) = -1)
[051] 359: CLOSE #1
[HA1] 360: END SUB
[000] 361:
[AS1] 362: DEFINT J, N
[0V6] 363: SUB Hiscore.Sort (Sort() AS Hiscore, nr)
[SU2] 364: FOR i = 1 TO nr - 1
[UV2] 365: FOR j = i + 1 TO nr
[947] 366: IF Sort(i).Versuche > Sort(j).Versuche TH
EN
[DM6] 367: SWAP Sort(i).Versuche, Sort(j).Versuch
e
[HD5] 368: SWAP Sort(i).Namen, Sort(j).Namen
[801] 369: END IF
[ZC1] 370: NEXT j, i
[HA1] 371: END SUB
[000] 372:
[X81] 373: DEFINT M
[0X1] 374: DEF SNG I-J, N
[F88] 375: SUB Hiscore.Speichern (Speichern() AS Hiscor
e, max)
[CU3] 376: OPEN "o", #1, "HISCORE.DAT"
[PN0] 377: CLS
[VP0] 378: hell
[MR1] 379: LOCATE 12, 20
[3M6] 380: PRINT "Hiscore-Tabelle wird gespeichert"
[BE2] 381: FOR i = 1 TO max
[835] 382: PRINT #1, Speichern(i).Versuche
[SC4] 383: PRINT #1, Speichern(i).Namen
[ZS0] 384: NEXT
[Y41] 385: delay 1
[441] 386: normal
[PN0] 387: CLS
[HA1] 388: END SUB
[000] 389:
[NL1] 390: DEFINT R-S
[VD5] 391: SUB Loesung (Richtig(), Schluss)
[PL1] 392: LOCATE 1, 1
[VP0] 393: hell
[B83] 394: FOR i = 1 TO Schluss
[1V2] 395: PRINT Richtig(i);
[ZS0] 396: NEXT
[441] 397: normal
[HA1] 398: END SUB
[000] 399:

```



```

[HK1] 400: DEFINT I-J
[LY1] 401: DEFSNG M, R-S
[F22] 402: SUB Mainzeil
[726] 403: PLAY "o2 L8 c f g a a o3 c o2 a g a+ a g L
      4 f"
[NB3] 404: PLAY "p8 o3 l16 c d e f"
[HA1] 405: END SUB
[000] 406:
[MH1] 407: DEFSNG I-J
[DK1] 408: SUB Menue
[PN0] 409: CLS
[HP5] 410: LOCATE 7, 25: PRINT "Namen aufnehmen"
[ZU5] 411: LOCATE 9, 25: PRINT "Hiscore-Tabelle"
[DX4] 412: LOCATE 11, 25: PRINT "Superhirn"
[S14] 413: LOCATE 13, 25: PRINT "Hilfe"
[344] 414: LOCATE 15, 25: PRINT "E N D E"
[0H1] 415: intensiv
[ZB5] 416: LOCATE 17, 25: PRINT "... wählen Sie"
[VP0] 417: hell
[FM6] 418: LOCATE 5, 20: PRINT "          M E N
      U E"
[WA3] 419: LOCATE 7, 20: PRINT " 1 "
[ZA3] 420: LOCATE 9, 20: PRINT " 2 "
[UB3] 421: LOCATE 11, 20: PRINT " 3 "
[ZB3] 422: LOCATE 13, 20: PRINT " 4 "
[4C3] 423: LOCATE 15, 20: PRINT " 5 "
[441] 424: normal
[HA1] 425: END SUB
[000] 426:
[N94] 427: FUNCTION Mitte$(Text$, x$)
[GE4] 428: rest = INT((80 - LEN(Text$)) / 2)
[MZ7] 429: Mitte$ = STRING$(rest, x$) + Text$ + STRIN
      GS(rest, x$)
      END FUNCTION
[R52] 430:
[000] 431:
[WD2] 432: DEFINT I-J, X-Y
[1M5] 433: SUB NamenAufnehmen (Spielername$)
[PN0] 434: CLS
[6M0] 435: x = 1
[272] 436: FOR y = 1 TO 11
[GP5] 437: CALL klein.Rahmen(x, y, 80 - x, 24 - y)
[231] 438: x = x + 1
[ZS0] 439: NEXT
[VM8] 440: Text$ = "* Bitte Namen eingeben * Bitte Na
      men eingeben"
[Z34] 441: CALL LaufSchrift(Text$, 1)
[7W6] 442: LOCATE 12, 17: PRINT STRING$(LEN(Text$) +
      1, " ")
[235] 443: LOCATE 12, 17: INPUT Spielername$
[HA1] 444: END SUB
[000] 445:
[D32] 446: DEFINT E, N, T
[M21] 447: DEFSNG J, X-Y
[AL6] 448: SUB Pruefen (Test(), Eingabe, n, Doppelt)
[XV1] 449: Doppelt = 0
[KF2] 450: FOR i = 1 TO n - 1
[E66] 451: IF Test(i) = Eingabe THEN 'Zahl doppelt
[NN1] 452: y = CSRLIN
[0G1] 453: x = POS(0)
[2X1] 454: Doppelt = -1
[221] 455: n = n - 1
[RN0] 456: BEEP
[SM1] 457: LOCATE 24, 1
[0H1] 458: intensiv
[E97] 459: PRINT Mitte(" Zahl doppelt eingegeben
      ", "-");
[441] 460: normal
[251] 461: delay 2
[7N1] 462: Fusszeile
[W36] 463: LOCATE y, x 'Cursor zurückstellen
[AD1] 464: EXIT FOR
[801] 465: END IF
[ZS0] 466: NEXT
[HA1] 467: END SUB
[000] 468:
[G21] 469: DEFINT M, R-S
[2D2] 470: DEFSNG E, I, N, T
[8U7] 471: SUB Spielen (Rechner(), Spieler(), max, Ver
      such)
[3V1] 472: Versuch = 0
[PN0] 473: CLS
[BE2] 474: FOR i = 1 TO max
[UP1] 475: PRINT " ";
[ZS0] 476: NEXT
[3K6] 477: LOCATE 1, 60: PRINT "Datum: "; Datum$(DATE$
      )
[P95] 478: LOCATE 2, 60: PRINT "Zeit : "; TIME$
[000] 479:
[7N1] 480: Fusszeile
[A44] 481: LOCATE 3, 1: PRINT "LfdNr: ";
[8D4] 482: PRINT TAB(50); "Platz", "Zahl"
[SL1] 483: LOCATE 3, 1
[S16] 484: CALL Zahlen.Erzeugen(Rechner(), max)
[000] 485:

```

```

[NB0] 486: DO
[KC3] 487: Versuch = Versuch + 1
[323] 488: PRINT Versuch; ". ";
[NN1] 489: y = CSRLIN
[WS5] 490: CALL Zahl.Aufnehmen(Spieler(), max)
[159] 491: CALL Ueberpruefen(Rechner(), Spieler(), m
      ax, Zahl, Platz)
[8X4] 492: LOCATE y, 50: PRINT Platz, Zahl
[KZ2] 493: IF Platz = max THEN
[ZC4] 494: '----- Lösung
[CM4] 495: CALL Loesung(Rechner(), max)
[C31] 496: blinkt
[RN0] 497: BEEP
[LC6] 498: LOCATE 1, 20: PRINT " ##### Lösung
      #####";
[251] 499: delay 2
[N31] 500: weiter
[K94] 501: CALL Auswertung(Versuch)
[TE1] 502: EXIT SUB
[801] 503: END IF
[K44] 504: LOOP UNTIL (Versuch >= 18)
[0H1] 505: intensiv
[C31] 506: blinkt
[RN0] 507: BEEP
[6V6] 508: LOCATE 1, 20: PRINT "##### A B B R U
      C H #####";
[395] 509: LOCATE 2, 1: PRINT "^^ Lösung ^^"
[441] 510: normal
[CM4] 511: CALL Loesung(Rechner(), max)
[251] 512: delay 2
[N31] 513: weiter
[K94] 514: CALL Auswertung(Versuch)
[HA1] 515: END SUB
[000] 516:
[XX1] 517: DEFINT I-J, N
[081] 518: DEFSNG M
[AV8] 519: SUB Ueberpruefen (Rechner(), Spieler(), n,
      Zahl, Platz)
[AW2] 520: Zahl = 0: Platz = 0
[K32] 521: FOR i = 1 TO n
[XC7] 522: IF Spieler(i) = Rechner(i) THEN Platz = P
      latz + 1
[L35] 523: ' Zahl und Platz richtig -----^
[M32] 524: FOR j = 1 TO n
[N27] 525: IF Spieler(i) = Rechner(j) THEN Zahl = Z
      ahl + 1
[TH5] 526: ' nur Zahl richtig Platz falsch --^
[ZC1] 527: NEXT j, i
[HA1] 528: END SUB
[000] 529:
[X81] 530: DEFINT M
[MY1] 531: DEFSNG N, R-S
[QG7] 532: SUB Zahl.Aufnehmen (Spieler() AS INTEGER, m
      ax)
[6L0] 533: i = 0
[NB0] 534: DO
[NN1] 535: y = CSRLIN
[0G1] 536: x = POS(0)
[A42] 537: CALL taste(x$)
[000] 538:
[F12] 539: IF x$ = "#" THEN
[BS5] 540: IF x = 6 THEN ' verhindert, daß x<10
[TM4] 541: x = 10 ' wird beim Löschen
[TF6] 542: SOUND 37, 10 ' tiefer Ton -> falsche
      Eingabe
[801] 543: END IF
[F14] 544: LOCATE y, x - 2: PRINT "X";
[EN4] 545: ' löscht ein Zeichen --^
[472] 546: LOCATE y, x - 4
[603] 547: IF i <> 0 THEN i = i - 1
[000] 548:
[FX5] 549: ELSEIF (i = max) AND (x$ = CHR$(13)) THEN
[3R3] 550: ' <Return>-Taste--^
[YF2] 551: PRINT "<ENTER>";
[TE1] 552: EXIT SUB
[000] 553:
[K46] 554: ELSEIF ASC(x$) >= 48 AND ASC(x$) <= 57 T
      HEN
[2L4] 555: ' 1 bis 0 -----^
[611] 556: i = i + 1
[4L2] 557: IF i = max + 1 THEN
[YF2] 558: PRINT "<ENTER>";
[TE1] 559: EXIT SUB
[801] 560: END IF
[323] 561: Spieler(i) = VAL(x$)
[A17] 562: CALL Pruefen(Spieler(), Spieler(i), i,
      Test)
[SP2] 563: IF Test <> -1 THEN
[145] 564: LOCATE y, x + 1: PRINT Spieler(i);
[801] 565: END IF
[000] 566:
[726] 567: ELSEIF i = max THEN PRINT "<ENTER>": EX
      IT SUB
[2A5] 568: ' ^-- max-Zahlen aufgenommen

```

Listing 41. »super4.bas« ist die endgültige Version von Superhirn


```

[000] 569:
[2R0] 570: ELSE
[D26] 571: y = CSRLIN - 1 'Y-Position speichern
[RU5] 572: x = POS(0) 'X-Position speichern
[VP0] 573: hell
[E28] 574: LOCATE 24, 1: PRINT "Fehler -> Keine Be-
dingung erfüllt";
[HB3] 575: PRINT STRING$(20, " ");
[591] 576: heulen 2
[7N1] 577: Fusszeile
[036] 578: LOCATE y + 1, x 'Koordinaten übergebe
n
[801] 579: END IF
[000] 580:
[000] 581:
[A73] 582: LOOP UNTIL (i > max)
[NG5] 583: ' max+1. Zeichen Abbruchkrit.
[HA1] 584: END SUB
[000] 585:
[Y81] 586: DEFINT N
[081] 587: DEFSG M
[823] 588: SUB Zahl.holen (x$)
[NB0] 589: DO
[TU3] 590: LOCATE 2, 68: PRINT TIMES$
[WM1] 591: x$ = INKEY$
[ET2] 592: LOOP UNTIL x$ <> ""
[HA1] 593: END SUB
[000] 594:
[6A7] 595: SUB Zahlen.Erzeugen (Super() AS INTEGER, n)
[K32] 596: FOR i = 1 TO n
[YB3] 597: Super(i) = INT(RND * 9)
[ZS0] 598: NEXT
[ZL3] 599: 'Zahlen überprüfen
[GX0] 600: PRINT
[NB0] 601: DO
[U52] 602: nocheinmal = 0
[K32] 603: FOR i = 1 TO n
[KH2] 604: FOR j = i + 1 TO n
[U74] 605: IF Super(i) = Super(j) THEN
[NB2] 606: nocheinmal = -1
[2C3] 607: Super(j) = INT(RND * 9)
[801] 608: END IF
[ZC1] 609: NEXT j, i
[PF4] 610: LOOP UNTIL (nocheinmal <> -1)
[GM1] 611: LOCATE 3, 7
[K32] 612: FOR i = 1 TO n
[G03] 613: PRINT Super(i); " ";
[6S2] 614: ' Programmtest-^
[ZS0] 615: NEXT
[GX0] 616: PRINT
[HA1] 617: END SUB
[000] 618:

```

Listing 41. »super4.bas« ist die endgültige Version von Superhirn

```

[H63] 1: DECLARE SUB Menue ()
[SE3] 2: DECLARE SUB Taste (x$)
[RC3] 3: DECLARE SUB Weiter ()
[Z23] 4: DECLARE SUB Hell ()
[W53] 5: DECLARE SUB hilfe ()
[4N6] 6: DECLARE SUB LaufSchrift (Text$, Anzahl!)
[4X3] 7: DECLARE SUB intensiv ()
[EC3] 8: DECLARE SUB blinkt ()
[6D3] 9: DECLARE SUB normal ()
[RN5] 10: DECLARE SUB Rahmen (x1%, y1%, x2%, y2%)
[2W5] 11: DECLARE SUB Loesche (x1%, y1%, x2%, y2%)
[414] 12: DECLARE SUB delay (Zeit!)
[000] 13:
[743] 14: 'Datei : PROG-UM.BAS
[M82] 15: 'Stand : 01.12.88
[ZD2] 16: 'Update: 03.12.88
[AV3] 17: 'Autor : Gerd Kebschull
[E20] 18:
[H64] 19: ' C by DOS-International
[000] 20:
[8P5] 21: '*****
*****
[UF4] 22: '***
**
[S37] 23: '***
Allgemeine Programmumgebung
**
[UF4] 24: '***
**
[8P5] 25: '*****
*****
[000] 26:
[TT1] 27: SUB blinkt
[CL1] 28: COLOR 16, 1
[HA1] 29: END SUB
[000] 30:
[ZL2] 31: SUB delay (Zeit)
[3Y2] 32: Ende = TIMER + Zeit
[NB0] 33: DO

```

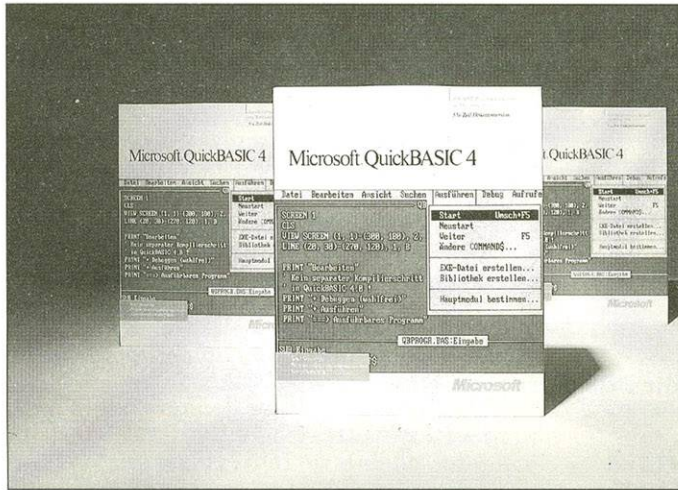
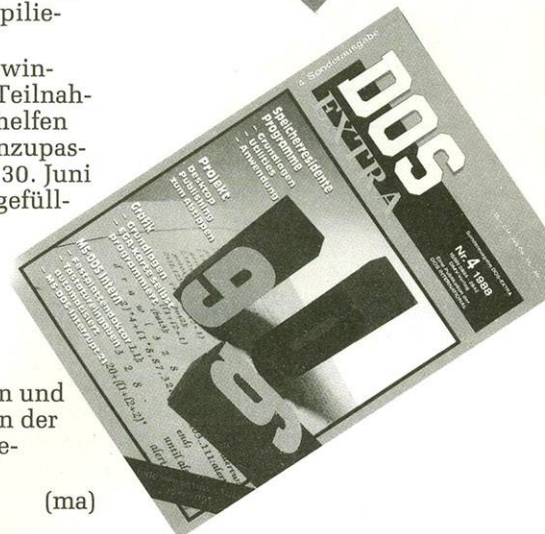
```

[0U3] 34: LOOP UNTIL TIMER > Ende
[HA1] 35: END SUB
[000] 36:
[8D1] 37: SUB Hell
[9N1] 38: COLOR 8, 15
[HA1] 39: END SUB
[000] 40:
[D72] 41: SUB intensiv
[2L1] 42: COLOR 10, 0
[HA1] 43: END SUB
[000] 44:
[DM1] 45: DEFINT X-Y
[C55] 46: SUB klein.Rahmen (x1, y1, x2, y2)
[4X5] 47: LOCATE y1, x1: PRINT " " 'ALT 218
[5T5] 48: LOCATE y1, x2: PRINT " " 'ALT 191
[AT5] 49: LOCATE y2, x1: PRINT " " 'ALT 192
[SV5] 50: LOCATE y2, x2: PRINT " " 'ALT 217
[FE8] 51: LOCATE y1, x1 + 1: PRINT STRING$(x2 - x1 - 1, "-") 'ALT 196
[5X6] 52: LOCATE y2, x1 + 1: PRINT STRING$(x2 - x1 - 1, "-")
[000] 53:
[A53] 54: FOR y = y1 + 1 TO y2 - 1
[ZV5] 55: LOCATE y, x1: PRINT " " 'ALT 179
[3X3] 56: LOCATE y, x2: PRINT " "
[ZS0] 57: NEXT
[HA1] 58: END SUB
[000] 59:
[HK1] 60: DEFINT I-J
[C81] 61: DEFSG Y
[0A5] 62: SUB LaufSchrift (Text$, Anzahl)
[000] 63:
[DY2] 64: Laenge = LEN(Text$)
[6P8] 65: x = INT(40 - Laenge / 2) - 1 'x-Koordinat
e für LaufText
[CN3] 66: x2 = 40 + INT(Laenge / 2) + 1
[0Z3] 67: CALL Loesche(x, 11, x2, 13)
[DH3] 68: CALL Rahmen(x, 11, x2, 13)
[H74] 69: LOCATE 12, x + 1: PRINT Text$
[933] 70: FOR j = 1 TO Anzahl
[U13] 71: FOR i = 1 TO Laenge
[637] 72: Druck$ = MID$(Text$, i, Laenge) + LEFT$(Text$, i)
[EL1] 73: a$ = INKEY$
[GG3] 74: IF a$ <> "" THEN EXIT SUB
[795] 75: '----- Abbruch bei Tastendruck
[NC4] 76: LOCATE 12, x + 1: PRINT Druck$
[461] 77: delay .2
[YC1] 78: NEXT i, j
[HA1] 79: END SUB
[000] 80:
[991] 81: DEFINT Y
[MH1] 82: DEFSG I-J
[6A4] 83: SUB Loesche (x1, y1, x2, y2)
[MD2] 84: FOR y = y1 TO y2
[SB6] 85: LOCATE y, x1: PRINT STRING$(x2 - x1 + 1, " ")
[ZS0] 86: NEXT
[HA1] 87: END SUB
[000] 88:
[GL1] 89: DEFSG X-Y
[HU1] 90: SUB normal
[HE1] 91: COLOR 7, 1
[HA1] 92: END SUB
[000] 93:
[DM1] 94: DEFINT X-Y
[7Z3] 95: SUB Rahmen (x1, y1, x2, y2)
[YK3] 96: LOCATE y1, x1: PRINT " "
[1W3] 97: LOCATE y1, x2: PRINT " "
[NX3] 98: LOCATE y2, x1: PRINT " "
[7W3] 99: LOCATE y2, x2: PRINT " "
[ZX6] 100: LOCATE y1, x1 + 1: PRINT STRING$(x2 - x1 - 1, "-")
[1Y6] 101: LOCATE y2, x1 + 1: PRINT STRING$(x2 - x1 - 1, "-")
[000] 102:
[A53] 103: FOR y = y1 + 1 TO y2 - 1
[RX3] 104: LOCATE y, x1: PRINT " "
[5X3] 105: LOCATE y, x2: PRINT " "
[ZS0] 106: NEXT
[HA1] 107: END SUB
[000] 108:
[GL1] 109: DEFSG X-Y
[X32] 110: SUB Taste (x$)
[NB0] 111: DO
[WM1] 112: x$ = INKEY$
[ET2] 113: LOOP UNTIL x$ <> ""
[HA1] 114: END SUB
[000] 115:
[ZU1] 116: SUB Weiter
[VP0] 117: Hell
[7U7] 118: LOCATE 24, 1: PRINT "... weiter nach Taste
ndruck";
[0C3] 119: PRINT STRING$(53, " ");
[441] 120: normal
[NB0] 121: DO
[EL1] 122: a$ = INKEY$
[RS2] 123: LOOP UNTIL a$ <> ""
[HA1] 124: END SUB

```

Listing 42. Die Fassung von »prog-um.bas« enthält die Programmumgebung von Superhirn

Sie



DOS Extra ist die dreimonatliche Sonderausgabe der DOS International. Sie greift Themen aus dem Stammheft auf und vertieft sie. Mit dieser Ausgabe liegt inzwischen das sechste DOS Extra vor. Jetzt möchten wir Sie etwas genauer kennenlernen und sind neugierig auf Ihre Meinung, um die künftigen DOS-Extra-Ausgaben Ihren Wünschen entsprechend zu gestalten.

Beantworten Sie dazu bitte die untenstehenden Fragen. Natürlich soll Ihre Mühe nicht umsonst sein! Als »Danke schön« für Ihre Mitarbeit verlosen wir unter allen Einsendern drei Exemplare von Microsofts Basic-Compiler Quick Basic 4.0 in der deutschen Version. Der Quick-Basic-Compiler glänzt nicht nur durch komfortable Bedienung und schnelle Compile, sondern auch durch seine besondere Arbeitsweise, die Basic-Programme bereits bei der Eingabe übersetzt, so daß ein gesonderter Compilierungsvorgang entfällt.

Doch sollten Sie auch nicht zu den glücklichen Gewinnern zählen, so profitieren Sie dennoch von Ihrer Teilnahme an dieser Umfrage, denn mit Ihren Antworten helfen Sie uns, das DOS Extra Ihren Erwartungen exakt anzupassen. Aus diesem Grund schicken Sie bitte bis zum 30. Juni 1989 (es gilt das Datum des Poststempels) den ausgefüllten Fragebogen zurück an:

Redaktionsbüro Everts & Hagedorn
Stichwort DOS Extra
Hans-Stießberger-Str. 3
8013 Haar/München

Mitarbeiter des Redaktionsbüros Everts & Hagedorn und des DMV-Verlags sowie deren Angehörige sind von der Teilnahme ausgenommen. Der Rechtsweg ist ausgeschlossen.

(ma)

Wie haben Ihnen die bisherigen DOS-Extra-Ausgaben gefallen?

- ☐ sehr gut
- ☐ gut
- ☐ mittel
- ☐ weniger gut
- ☐ gar nicht

Wie viele DOS-Extra-Ausgaben (außer diesem Heft) besitzen Sie bereits?

- ☐ keine
- ☐ eine
- ☐ zwei
- ☐ drei
- ☐ vier
- ☐ fünf

Welche Erscheinungsweise für die DOS Extra ziehen Sie vor?

- ☐ monatlich
- ☐ zweimonatlich
- ☐ dreimonatlich
- ☐ viermonatlich
- ☐ halbjährlich

Möchten Sie im DOS Extra lieber mehrere Themen oder ein einzelnes Thema ausführlich behandelt sehen?

- ☐ mehrere Themen
- ☐ einzelnes Thema

Welche Themen?

Welche Programmiersprachen besitzen Sie sonst noch?

- ☐ Turbo Pascal 3.0
- ☐ Turbo Pascal 4.0
- ☐ Turbo Pascal 5.0
- ☐ Quick C 1.0
- ☐ Quick C 2.0
- ☐ Turbo C 1.0
- ☐ Turbo C 1.5
- ☐ Turbo C 2.0
- ☐ MASM 4.0
- ☐ MASM 5.0
- ☐ TASM

Mit welcher Grafik ist Ihr Computer ausgerüstet?

- ☐ Monochromadapter
- ☐ Hercules-Grafik
- ☐ CGA-Farbgrafik
- ☐ EGA-Grafik
- ☐ VGA-Grafik

Welche Computer-Zeitschriften lesen Sie?

- ☐ Chip
- ☐ Chip Tool
- ☐ Computer Persönlich
- ☐ C't
- ☐ Data Welt
- ☐ DOS International
- ☐ Happy Computer
- ☐ mc
- ☐ PC Magazin
- ☐ PC Magazin Plus
- ☐ PC Pur
- ☐ Toolbox
- ☐ Window

Ich erkläre mich damit einverstanden, daß die hier gemachten Angaben elektronisch ausgewertet werden. Das Redaktionsbüro Everts & Hagedorn verpflichtet sich, die persönlichen Daten vertraulich zu behandeln, nicht weiterzugeben und nur für die Ermittlung der Gewinner zu verwenden.

Vorname, Name

Straße

PLZ, Wohnort

Welchen Basic-Interpreter oder Basic-Compiler besitzen Sie?

- ☐ GW-Basic
- ☐ Quick Basic 2.0
- ☐ Quick Basic 3.0
- ☐ Quick Basic 4.0
- ☐ Quick Basic 4.5
- ☐ Turbo Basic
- ☐ True Basic
- ☐ Better Basic

Zum Thema Listing

Ohne Spielregeln geht es nicht

Wenn Sie mit dem Gedanken spielen, uns ein Listing anzubieten, oder wenn Sie zum Kreis derer gehören, deren Angebot bereits einmal dankend abgelehnt wurde, dann sollten Sie jetzt weiterlesen. Sie erfahren hier nämlich, welche Dinge zu beachten sind, um die Chance für einen Abdruck Ihres Listings entscheidend zu erhöhen.

Listings in den Rubriken »Anwendung«, »Utility« und »Tips & Tricks« sind dazu gedacht, Ihnen Anregungen zu geben, wie Sie alltägliche oder auch spezielle Aufgabenstellungen und Probleme programmtechnisch in den Griff bekommen, welche Verfahren anzuwenden sind, und wie Sie dabei am besten vorgehen.

Gerade in den beiden erstgenannten Rubriken haben die Listings eine doppelte Funktion: Zum einen stellen sie eine Variante zur Lösung vorgegebener Aufgabenstellungen dar. Andererseits nutzen viele Leser die Programme nicht in ihrer Gesamtheit, sondern greifen einzelne Programmteile, Funktionen und Prozeduren auf, um sie – in leicht abgewandelter Form – in eigenen Applikationen anzuwenden. Dies gelingt um so leichter, je mehr Informationen und Hinweise Sie zu den einzelnen Programmteilen erhalten.

Leider erhalten wir viel zu oft Einsendungen in der »Vogel-friß-oder-stirb-Form«: »Beiliegend erhalten Sie mein Programm »XYirgendwie«. Viel Spaß!« Zu diesem informativen Schreiben (nicht selten ohne Absender) ereilt uns der Spaß in Form eines wandfüllenden Listingausdrucks oder einer Diskette, die nichts weiter enthält als den Quellcode eines Programms in einer beliebigen Programmiersprache. Für welche Version und welchen Compiler und vor allem zu welchem Zweck die Software geschrieben wurde, bleibt für immer das Geheimnis des Autors.

Liebe Programmautoren, Redakteure sind leider (Gott sei Dank?) keine wundersamen Wesen, die in Sekundenschnelle ergründen, was Ihre Gehirne in nächtelanger Arbeit ausgebrütet haben. Geben Sie uns daher eine Chance, das Resultat Ihrer geistigen Anstrengungen richtig zu beurteilen, indem Sie die folgenden Regeln beherzigen.

Ihre Einsendung sollte sich aus drei Teilen zusammensetzen:

- dem Anschreiben,
- Ihrem Programm auf einer mit 360 KByte formatierten Diskette und
- einer Programmbeschreibung (als reine ASCII-Datei ebenfalls auf der Diskette).

Auf dem Anschreiben und der Diskette geben Sie bitte Ihre vollständige Adresse und Ihre Telefonnummer an; Rückfragen der Redaktion lassen sich so am schnellsten abklären. Führen Sie des weiteren alle auf der Diskette befindlichen Dateien namentlich auf und geben Sie stichpunktartig an, wozu die jeweilige Datei gedacht ist beziehungsweise welchen Inhalt sie hat.

Schließlich benötigen wir noch Angaben über den Sinn und Zweck Ihres Programms, wie man es startet (eventuell zu übergebende Parameter, Hardware-Voraussetzungen und so weiter) und für welche Compiler-Version es geschrieben wurde.

Ihr Programm sollten Sie uns in zweifacher Ausführung auf einer mit 360 KByte formatierten Diskette zukommen lassen: den Quellcode als reine ASCII-Datei und daneben das ausführbare Programm in Form einer Exe- oder Com-Datei. Bei Basic-Listings, die für einen Basic-Interpreter (beispielsweise GW-Basic) gedacht sind, ist die zweite Form natürlich hinfällig. Denken Sie in diesem Zusammenhang bitte daran, beim Speichern der Basic-Programme den Parameter »a« anzuhängen, um den Quellcode als ASCII-Datei zu sichern. Zum Beispiel:

```
save "prog",a
```

Ein sehr wichtiger Punkt ist die maximale Spaltenzahl (Anzahl der Zeichen pro Zeile) Ihres Listings. Der Abdruck eines Listings in der DOS International sieht folgendermaßen aus: Links steht, von eckigen Klammern eingeschlossen, ein dreistelliger Prüfsummencode, es schließt sich die Zeilennummer an, gefolgt von einem Doppelpunkt, und erst dann kommt die Programmzeile. Für eine solche Programmzeile stehen daher nur noch 43 Spalten zur Verfügung. Alle darüber hinausreichenden Zeichen werden automatisch auf die nächste Zeile umgebrochen. Dadurch geht die Struktur von Programmen verloren (ganz gleich, ob es sich dabei um C-, Pascal-, Basic- oder Assembler-Listings handelt), sie werden unübersichtlich und schwerer lesbar.

Deshalb unsere dringende Bitte: Strukturieren Sie Ihre Programme un-

ter Berücksichtigung einer maximalen Zeilenlänge von 43 Zeichen. Bei Beachtung einiger weniger Punkte wird die Strukturierung Ihres Listings durch diese Vorgabe nicht beeinträchtigt.

- Rücken Sie den Quellcode pro Strukturblock (keine willkürlichen Einrückungen) um maximal zwei Zeichen ein.
- Plazieren Sie Kommentare in eigenen Zeilen, anstatt sie an eine Programmzeile anzuhängen.
- Brechen Sie lange Programmzeilen logisch um, das heißt so, daß die Struktur Ihres Programms sichtbar erhalten bleibt.

Eines ist dabei selbstverständlich: Trotz aller Manipulationen, die Sie an Ihrem Quelltext vornehmen, um der Vorgabe von 43 Zeichen pro Programmzeile zu entsprechen, muß dieses Listing immer noch der Syntax des verwendeten Compilers entsprechen. Mit anderen Worten: Es muß sich ohne (Syntax-) Fehlermeldungen compilieren lassen.

Klärende Worte erleichtern die Umformung von Programmteilen für die eigenen Bedürfnisse

Ein undokumentiertes Programm ist nur eine halbe Sache. Der Umstand, daß ein Programm dem interessierten Leser eine Fülle an Prozeduren und Funktionen für eigene Anwendungen beschert, macht eine ausführliche Dokumentation um so dringlicher. Was helfen die schönsten Programmauszüge, wenn man sie – mangels Information – nicht für die eigenen Zwecke anpassen und umformen kann.

Sie, als der Programmautor, wissen am besten über die Vorzüge, Besonderheiten, Feinheiten und eventuell auch Unzulänglichkeiten Ihres Programms Bescheid. Fassen Sie diese Informationen doch einfach in Form eines Artikels zusammen. Sie erleichtern uns die Arbeit, wenn Sie den Text nicht nur in gedruckter Form, sondern auch als reine ASCII-Datei (ohne irgendwelche Steuerzeichen) auf Diskette Ihrer Sendung beilegen. Vermeiden Sie dabei bitte in Ihrem Text durchgängige Großschreibung, zum Beispiel bei Produktnamen oder Programmbefehlen.

Anregungen bezüglich Art und Stil solcher Artikel können Sie jeder DOS-Ausgabe entnehmen. Auch unsere Redakteure stehen Ihnen für Informationen in dieser Richtung gerne zur Verfügung.

In diesem Sinne: Wir freuen uns auf Ihr Listing!

(zi)

Die Kartenfrage

Bei der Programmierung eines Grafikprogramms, das auch auf unterschiedlichen Computern laufen soll, stellt sich die Frage nach der verwendeten Grafikkarte. Hier wird ein Diagnostik-Programm in GW-Basic vorgestellt, das darüber präzise Informationen liefert.

Die folgenden GW-Basic-Routinen (im Listing »chkcard.bas« zusammengefaßt) lösen das Problem der Grafikkartenerkennung.

Die Routine »adapter.bas« ab Zeile 2000 ist die eigentliche Routine zur Erkennung des Bildschirmadapters. Mit dem Programmteil »interbas.bas« ab Programmzeile 1000 greift GW-Basic auf Interrupts zu. Diese Funktion benötigt das Listing »adapter.bas«, weil BIOS-Interrupts aufgerufen werden müssen. »adapter.bas« unterscheidet zwischen einem Monochromadapter, einer CGA-, EGA- und VGA-Karte. Der Vollständigkeit halber ist auch die Routine »findherc.bas« ab Zeile 3000 enthalten, die feststellt, ob ein Hercules-Grafikadapter installiert ist oder nicht. Da dieser Test aufwendig ist und mit GW-Basic zirka zwei Minuten dauert, empfiehlt es sich nicht, ihn mit dem Interpreter durchzuführen. Bei der Compilierung (beispielsweise mit SBasic aus dieser DOS-Extra-Ausgabe) reduziert sich diese Zeit auf einen Sekundenbruchteil, so daß eine Anwendung hier schon sinnvoller erscheint.

Die Adapterroutine liefert mehrere Werte in verschiedenen Variablen zurück. In der Variablen »monitorseg« steht das Segment, in dem der Bildschirmspeicher beginnt. Dabei bedeutet ein Wert von B800hex, daß ein Farbgrafikadapter installiert ist, ein Wert von B000hex, daß ein Monochrom- oder ein Hercules-Adapter Verwendung findet. Die Variable »config« liefert die Konfiguration des jeweiligen Rechners, die Aufschlüsselung finden Sie im Programm selbst ab Zeile 2090. Die Variable »speicher« gibt an, wieviel KByte Bildschirmspeicher auf der EGA-

beziehungsweise VGA-Karte installiert sind. Zusätzlich steht in der Variablen »modus« der aktuelle Bildschirmmodus des Systems (Tabelle) und in der Variablen »spalten« die gerade angezeigte Anzahl der Spalten auf dem Bildschirm (40 oder 80).

Nun zur Funktion der Routine »adapter.bas«. Die Routine beginnt damit, festzustellen, ob eine Farbgrafik- oder eine Monochromkarte installiert ist. Wenn die Bits 4 und 5 des BIOS-Konfigurationsregister 0040:0010hex gesetzt sind, also den Wert 3 enthalten, ist eine Monochrom-, anderenfalls eine Farbgrafikkarte installiert.

Nun wird geprüft, ob eine VGA-Karte vorhanden ist. Die Routine ruft eine Funktion auf, die nur im VGA-BIOS implementiert ist. Wenn nach diesem Aufruf das AH-Register immer noch den gleichen Wert aufweist, heißt das, daß die BIOS-Funktion nicht implementiert ist und folglich keine VGA-Karte vorhanden sein kann.

Dann folgt der Test auf Vorhandensein einer EGA-Karte. Dafür wird die Funktion 12hex des BIOS-Videointerrupt 10hex aufgerufen. Diese Funktion besitzt zwei Unterfunktionen, deren Nummer im BL-Register übergeben wird. Wenn das BL-Register gleich 10hex ist, wird eine Funktion aktiviert, die EGA-Statusinformationen liefert. Diese Funktion ist interessant für unsere Routine.

Ist eine CGA-Karte vorhanden, hat BL nach dem Aufruf immer noch den Wert 10hex, weil diese Funktion des BIOS-Video-Interrupt nur im EGA- oder VGA-BIOS implementiert ist. Ist eine EGA- oder VGA-Karte vorhanden, hat BL nach dem Interruptaufruf einen Wert von 0, 1, 2 oder 3, je nach Bildschirmspeicher, es nimmt jedoch nie den Wert 10hex an. Somit ist die Unterscheidung einfach: Weist BL bei Rückkehr den Wert 10hex auf, ist eine CGA-Karte installiert, ansonsten enthält BL die Größe des Bildschirmspeichers der EGA- oder VGA-Karte.

Jetzt bleibt nur noch festzustellen, was für ein Monitor angeschlossen ist. Darüber gibt derjenige Wert Auskunft, der sich nach dem Interruptaufruf im BH-Register befindet. Ein Wert von 0 bedeutet, daß ein Farbmonitor, ein Wert von 1, daß ein Monochrom-Monitor angeschlossen ist.

Der Videomodus für die Funktionen 00, 0F, 10 und 11 des Videointerrupts 10

Videomodus	Text/Grafik	Grafikauflösung	Textauflösung	Farben	Bildschirmseiten	Anmerkung
00	Text	–	40x25	16 Graustufen	8	–
01	Text	–	40x25	16	8	–
02	Text	–	80x25	16 Graustufen	4	Standardwert für CGA-, EGA- und VGA-Karte
03	Text	–	80x25	16	4	–
04	Grafik	320x200	–	4	1	–
05	Grafik	320x200	–	4 Graustufen	1	–
06	Grafik	640x200	–	monochrom	1	–
07	Text	–	80x25	monochrom	1	Standardwert für Monochromadapter und Hercules-Grafikkarte
08	Grafik	160x200	–	16	2	nur auf IBM-PC-Junior
09	Grafik	320x200	–	16	1	nur auf IBM-PC-Junior
10	Grafik	640x200	–	4	2	nur auf IBM-PC-Junior
11						Reserviert
12						Reserviert
13	Grafik	320x200	–	16	4	nur EGA- und VGA-Karte
14	Grafik	640x200	–	16	2	nur EGA- und VGA-Karte
15	Grafik	640x350	–	monochrom	1	nur EGA- und VGA-Karte
16	Grafik	640x350	–	16	1	nur EGA- und VGA-Karte
17	Grafik	640x480	–	2	1	nur VGA-Karte
18	Grafik	640x480	–	16	1	nur VGA-Karte
19	Grafik	320x200	–	256	1	nur VGA-Karte

Tabelle. Die verschiedenen Videomodi des PC


```

Ihre Grafikkarte wird mit einem Monochrom-Monitor betrieben.
Eine Hercules-Grafikkarte ist nicht vorhanden.
Eine CGA-Grafikkarte ist nicht vorhanden.
Grafikkonfiguration: EGA-Karte und digitaler Monochrom-Monitor
Die Grafikkarte besitzt 256 KByte Bildschirmspeicher.
Die Karte befindet sich in einem monochromen Modus.
Zur Zeit ist der Videomodus 7 aktiviert.
Auf dem Bildschirm werden 88 Spalten angezeigt.
A:\>

```

»chkcard.bas« zeigt alle Parameter einer Grafikkarte

Im folgenden wird noch die Funktion 0Fhex des BIOS-Videointerrupt aufgerufen, die den aktuellen Bildschirmstatus liefert. Im AH-Register steht anschließend die Anzahl der Spalten und im AL-Register der Bildschirmmodus. Dabei ist allerdings zu beachten, daß die Nummer des Bildschirmmodus, die das BIOS zurückliefert, eine andere ist als die, die Basic nach dem Screen-Befehl erwartet. Soll bei Programmende der vorherige Bildschirmmodus wiederhergestellt werden, so benutzen Sie am besten ebenfalls einen BIOS-Aufruf, nämlich die Funktion 0hex des BIOS-Video-Interrupt.

(Lutz Lennemann/tr)

Name	chkcard.bas
Sprache	GW-Basic, SBasic
Grafikkarte	jede
Besonderheiten	keine

```

[529] 10 'Dieses Programm untersucht die Grafikkarte Ih
res PC
[KF8] 20 'und gibt alle relevanten Informationen zurück
[M57] 30 '(C) 1989 Redaktionsbüro Everts&Hagedorn
[7G3] 40 'DOS International
[1D0] 50 '
[1W9] 60 'Maschinenprogramm für Interruptaufrufe initia
lisieren
[XV1] 70 DEFINT A-Z
[7L1] 80 RESTORE
[3VD] 90 READ CODE$: IF CODE$<>"ENDE" THEN INTERRUPT$=
INTERRUPT$+CHR$(VAL("&H"+CODE$)): GOTO 90
[KR1] 100 DEF SEG
[UA3] 110 STAPEL$=SPACE$(128)
[2C7] 120 AH=0: AL=0: BH=0: BL=0: CH=0: CL=0: DH=
0: DL=0
[P64] 130 RUECKGABEPARAMETER=255
[Z2E] 140 PRINT "Bitte warten ... (Test dauert bei eine
m 4,77-MHz-PC mit GW-Basic zwei Minuten)"
[1G7] 150 'Typ des Bildschirmadapters feststellen
[CY1] 160 GOSUB 2000
[MY7] 170 'Feststellen, ob Herculeskarte vorhanden ist
[PY1] 180 GOSUB 3000
[351] 190 CLS
[F57] 200 PRINT "Ihre Grafikkarte wird mit einem ";
[WBG] 210 IF MONITORSEG=&HB000 THEN PRINT "Monochrom-Mo
nitor betrieben." ELSE PRINT "Farbmonitor betrieb
en."
[EA8] 220 PRINT: PRINT "Eine Hercules-Grafikkarte ist
";
[BV5] 230 IF HERCULES=0 THEN PRINT "nicht ";
[FA3] 240 PRINT "vorhanden."
[R2G] 250 IF HERCULES=1 THEN PRINT: PRINT "Die Hercule
s-Grafikkarte besitzt 64 KByte Bildschirmspeicher
"
[9C7] 260 PRINT: PRINT "Eine CGA-Grafikkarte ist ";
[F7C] 270 IF KARTE$="CGA/Hercules" AND MONITORSEG=&HB00
0 THEN KARTE$="CGA": GOTO 290
[LT2] 280 PRINT "nicht ";
[YA3] 290 PRINT "vorhanden."
[DCG] 300 IF KARTE$="CGA" THEN PRINT: PRINT "Die CGA-F
arbgrafikkarte besitzt 16 KByte Bildschirmspeiche
r."

```

Listing. »chkcard.bas«, Systemanalyse für Grafikkarten

Fraktal Generator 3D

Neu:
Die Dritte
Dimension

Meisterstücke der Computergrafik

High-Speed

Höchsteffiziente Programmierung
Nur noch 7 Sekunden für das "Apfelmännchen" auf dem Amiga!

Mandelbrot- und Juliamenge

Glättungsfunktion

Super-Parallel-Projektion

Frei wählbarer horizontaler Blickwinkel mit 360 Grad:
Betrachten Sie das "Fraktalobjekt" von allen Seiten

Stufenloser vertikaler Blickwinkel:

Wahlweise Sicht von oben und unten, schräg oder in der Totalen

Voller Bedienungskomfort

Auswahl komplett mit Pulldown-Menüs
Wahlweise Steuerung mit Maus oder Tastatur

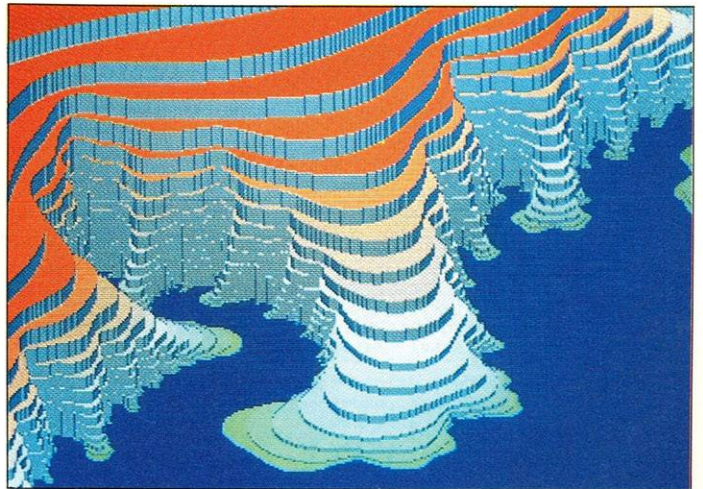
Mehrere separate Bildspeicher

Der Amiga speichert im IFF-Format ab. Die Bilder lassen sich dadurch in andere Malprogramme laden.

Phantastische Farbmöglichkeiten

Separate Farbzuoordnung für die einzelnen Bilder
Animationsmöglichkeit durch Color-Cycling

Die Farben lassen sich auch nachträglich beliebig verändern.
Gilt in beschränktem Maße für den Atari



PC 3D MS-DOS ab 2.0; PC-XT/AT mit EGA-Karte
oder: Amstrad/Schneider PC 1512.

Fraktal Generator 3D

MS-DOS 5 1/4"-Disk. Best.-Nr. 248
MS-DOS 3 1/2"-Disk. Best.-Nr. 249

69,- DM*

Amiga 3D

Commodore Amiga mit 512 KB

Amiga 3 1/2"-Disk. Best.-Nr. 2901

69,- DM*

Atari 3D

Atari ST

Atari 3 1/2"-Disk. Best.-Nr. 2951

69,- DM*

CPC 3D

CPC 464 / 664 / 6128

Kassette Best.-Nr. 210

3"-Disk. Best.-Nr. 211

59,- DM*

69,- DM*

* Unabhängig von der Anzahl der bestellten Programme berech-
nen wir für das Inland 3,- DM bzw. 5,- DM Porto/Verpackung.
– Unverbindliche Preisempfehlung –

Bitte Bestellkarte benutzen

DMV-Verlag · Postfach 250 · 3440 Eschwege


```

[4E3] 310 IF CONFIG=0 THEN 420
[TR6] 320 PRINT : PRINT "Grafikkonfiguration: ";
[8WC] 330 IF CONFIG=1 THEN PRINT "Monochromadapter und d
igitaler Monochrom-Monitor"
[16A] 340 IF CONFIG=2 THEN PRINT "CGA-Karte und digitale
r Farbmonitor"
[E6A] 350 IF CONFIG=4 THEN PRINT "EGA-Karte und digitale
r Farbmonitor"
[M9B] 360 IF CONFIG=5 THEN PRINT "EGA-Karte und digitale
r Monochrom-Monitor"
[0TB] 370 IF CONFIG=6 THEN PRINT "PGA-Karte und Professi
onal-Graphics-Monitor"
[F9B] 380 IF CONFIG=7 THEN PRINT "VGA-Karte und analoger
Monochrom-Monitor"
[32A] 390 IF CONFIG=8 THEN PRINT "VGA-Karte und analoger
Farbmonitor"
[5U5] 400 IF CONFIG>8 THEN PRINT "unbekannt"
[BC1] 410 PRINT
[BU3] 420 IF SPEICHER=0 THEN 460
[1H5] 430 PRINT "Die Grafikkarte besitzt";
[PS9] 440 IF SPEICHER=256 AND KARTE$="VGA" THEN PRINT "
mindestens";
[NM7] 450 PRINT SPEICHER;"KByte Bildschirmspeicher."
[T88] 460 PRINT : PRINT "Die Karte befindet sich in ein
em ";
[7XC] 470 ON MODUS+1 GOTO 480,490,480,490,490,480,480,4
80,490,490,490,490,490,490,480,490,490,490,490
[G76] 480 PRINT "monochromen Modus.": GOTO 500
[HB3] 490 PRINT "Farbmodus."
[BSA] 500 PRINT : PRINT "Zur Zeit ist der Videomodus";M
ODUS;"aktiviert."
[1NB] 510 PRINT : PRINT "Auf dem Bildschirm werden";SPA
LTEN;"Spalten angezeigt."
[431] 520 END
[RM0] 1000 '
[N17] 1010 ' Unterprogramm für Interrupt-Aufruf
[2M0] 1020 '
[6RB] 1030 INTERRUPT!=PEEK(VARPTR(INTERRUPT$)+2)*256+PE
EK(VARPTR(INTERRUPT$)+1)
[5LF] 1040 CALL INTERRUPT!(AH,AL,BH,BL,CH,CL,DH,DL,NUMM
ER,DES.INTERRUPTS,RUECKGABEPARAMETER,STAPEL$)
[CV1] 1050 RETURN
[TM0] 2000 '
[KM6] 2010 ' Diese Routine stellt fest, welcher
[KM5] 2020 ' Bildschirmadapter in einem PC
[F15] 2030 ' installiert ist und stellt
[4S6] 2040 ' zusätzliche Informationen bereit.
[DN0] 2050 '
[A05] 2060 ' MONITORSEG = Segment des
[375] 2070 ' Bildschirmspeichers, wobei
[FD7] 2080 ' Farbgrafik=B800hex und Monochrom=B000hex
[9A4] 2090 ' CONFIG= 0:kein Monitor
[AD7] 2100 ' 1: MDA + digitaler Monochrom-Monit
or
[L26] 2110 ' 2: CGA + digitaler Farbmonitor
[L36] 2120 ' 4: EGA + digitaler Farbmonitor
[XC7] 2130 ' 5: EGA + digitaler Monochrom-Monit
or
[MR7] 2140 ' 6: PGA + Professional-Graphics-Mon
itor
[127] 2150 ' 7: VGA + analoger Monochrom-Monito
r
[V06] 2160 ' 8: VGA + analoger Farbmonitor
[0Y5] 2170 ' KARTE$ = Typ der Grafikkarte
[X9B] 2180 ' SPEICHER = Größe des Bildschirmspeichers
auf der Grafikkarte
[S79] 2190 ' SPALTEN = Anzahl der angezeigten Spalte
n (40 oder 80)
[CS6] 2200 ' MODUS = Videomodus der Grafikkarte
[3N0] 2210 '
[6M5] 2220 ' BIOS-Datensegment aktivieren
[M34] 2230 DEF SEG=&H40 : CONFIG=0
[C15] 2240 ' Test auf farbig/monochrom:
[KZ8] 2250 ' Speicherstelle im BIOS-Datensegment untersu
chen
[RVB] 2260 IF (PEEK(&H10) AND &H30)=&H30 THEN MONITORS
E=&HB000 ELSE MONITORS E=&HB800
[EX5] 2270 ' Basic-Datensegment aktivieren
[4S1] 2280 DEF SEG
[6U2] 2290 ' Test auf VGA:
[A1B] 2300 ' Funktion 1Ahex, Unterfunktion 0 des Video-I
nterrupt aufrufen
[SH2] 2310 AH=&H1A : AL=0
[XA5] 2320 NUMMER.DES.INTERRUPTS=&H10
[R32] 2330 GOSUB 1030
[GC1] 2340 'VGA?
[626] 2350 IF AH<>&H1A THEN CONFIG=BL : KARTE$="VGA"
[CT2] 2360 'Test auf EGA:
[FPB] 2370 'Funktion 12hex, Unterfunktion 10hex des Vid
eo-Interrupt aufrufen
[5K2] 2380 AH=&H12 : AL=0
[1N2] 2390 BH=0 : BL=&H10
[SA5] 2400 NUMMER.DES.INTERRUPTS=&H10

```

```

[K32] 2410 GOSUB 1030
[AW3] 2420 'weder EGA noch VGA?
[WY7] 2430 IF BL=&H10 THEN KARTE$="CGA/Hercules": GOTO
2470
[FA1] 2440 'EGA?
[ZS7] 2450 IF KARTE$<>"VGA" THEN CONFIG=BH+4 : KARTE$="
EGA"
[FC3] 2460 SPEICHER=(BL+1)*64
[4M4] 2470 'Videomodus ermitteln:
[F28] 2480 'Funktion 0Fhex des Video-Interrupt aufrufen
[YF2] 2490 AH=&HF : AL=0
[VA5] 2500 NUMMER.DES.INTERRUPTS=&H10
[N32] 2510 GOSUB 1030
[654] 2520 MODUS=AL : SPALTEN=AH
[NV1] 2530 RETURN
[VM0] 3000 '
[C87] 3010 'Diese Routine prüft, ob eine Hercules-
[1D5] 3020 'Grafikkarte installiert ist.
[W46] 3030 'HERCULES=0: keine Hercules-Karte
[DX6] 3040 'HERCULES=1: Hercules-Karte vorhanden
[FN0] 3050 '
[LC2] 3060 HERCULES=0
[CH5] 3070 'Hercules Display Status Port
[0N3] 3080 STATUS.PORT=&H3BA
[2H4] 3090 'aktuellen Status lesen
[PH5] 3100 OLD.STATUS=INP(STATUS.PORT)
[M43] 3110 'Bit 7 isolieren
[CP5] 3120 OLD.STATUS=OLD.STATUS AND &H80
[3H6] 3130 '6400mal prüfen, ob sich Bit 7 ändert
[9V2] 3140 FOR I=1 TO 6400
[NG4] 3150 'aktuellen Status lesen
[AL5] 3160 NOW.STATUS=INP(STATUS.PORT)
[C53] 3170 'Bit 7 isolieren
[LU5] 3180 NOW.STATUS=NOW.STATUS AND &H80
[ZYA] 3190 'wenn Status sich ändert, ist es vielleicht
eine Hercules-Karte
[AU6] 3200 IF NOW.STATUS<>OLD.STATUS THEN I=6400
[E91] 3210 NEXT
[VUB] 3220 'wenn es keine Hercules-Karte sein kann, zur
ück zum Hauptprogramm
[9V6] 3230 IF NOW.STATUS=OLD.STATUS THEN RETURN
[HN0] 3240 '
[7WC] 3250 'Zweiter Test: Wenn ein 16-Bit-Wert im Bilds
chirmspeicher geändert und
[HDB] 3260 'wieder gelesen werden kann, müssen mindeste
ns 32 KByte Speicher
[E2E] 3270 'vorhanden sein, das heißt, eine Herculeskar
te ist tatsächlich installiert
[1P0] 3280 '
[SZ5] 3290 'Hercules Configuration Switch
[HF3] 3300 CONFIG.PORT=&H3BF
[Y27] 3310 'Grafikmodus freigeben, falls gesperrt
[4H3] 3320 OUT CONFIG.PORT,1
[TB7] 3330 'Bildschirmspeicher-Segment aktivieren
[RR2] 3340 DEF SEG=&HB000
[MT5] 3350 'niederwertiges Byte (LSB) lesen
[714] 3360 LOW.BYTE=PEEK(&H7FFE)
[CV5] 3370 'höherwertiges Byte (MSB) lesen
[X54] 3380 HIGH.BYTE=PEEK(&H7FFF)
[B68] 3390 '16-Bit-Wert 55AAhex in Speicherzelle schrei
ben
[XX2] 3400 POKE &H7FFE,&HAA
[YW2] 3410 POKE &H7FFF,&H55
[0K9] 3420 IF PEEK(&H7FFE)=&HAA AND PEEK(&H7FFF)=&H55 T
HEN HERCULES=1
[999] 3430 'Wenn sich der Speicher beschreiben ließ: He
rcules
[LU3] 3440 POKE &H7FFE,LOW.BYTE
[KX3] 3450 POKE &H7FFF,HIGH.BYTE
[LF7] 3460 'Basic-Speichersegment wieder aktivieren
[8S1] 3470 DEF SEG
[8W1] 3480 RETURN
[5R5] 4000 'Daten für Maschinenprogramm
[U75] 4010 DATA 55,8B,EC,8B,FC,8B,76,06,8A,04
[615] 4020 DATA B4,00,46,03,04,48,8B,E0,57,B8
[205] 4030 DATA 00,00,8E,D8,A1,90,00,50,A1,92
[MX4] 4040 DATA 00,50,E8,00,00,5B,81,C3,56,00
[W15] 4050 DATA 89,1E,90,00,0E,58,A3,92,00,8C
[785] 4060 DATA C0,8E,D8,8B,76,1A,8A,24,8B,76
[U35] 4070 DATA 18,8A,04,8B,76,16,8A,3C,8B,76
[S45] 4080 DATA 14,8A,1C,8B,76,12,8A,2C,8B,76
[265] 4090 DATA 10,8A,0C,8B,76,0E,8A,34,8B,76
[N15] 4100 DATA 0C,8A,14,50,53,8B,76,0A,8A,04
[W15] 4110 DATA E8,00,00,5B,81,C3,0E,00,2E,88
[255] 4120 DATA 07,EB,00,5B,58,55,CD,00,EB,18
[Z55] 4130 DATA 90,FB,8B,C7,B4,00,83,C4,12,5D
[N15] 4140 DATA 1F,07,83,C4,06,5D,8B,76,08,89
[155] 4150 DATA 04,EB,2A,90,5D,8B,76,1A,88,24
[G35] 4160 DATA 8B,76,18,88,04,8B,76,16,88,3C
[G35] 4170 DATA 8B,76,14,88,1C,8B,76,12,88,2C
[G25] 4180 DATA 8B,76,10,88,0C,8B,76,0E,88,34
[W35] 4190 DATA 8B,76,0C,88,14,B8,00,00,8E,D8
[805] 4200 DATA 58,A3,92,00,58,A3,90,00,8C,C0
[EL5] 4210 DATA 8E,D8,5F,8B,E7,5D,CA,16,00,ENDE

```

Listing. »chkcard.bas«, Systemanalyse für Grafikkarten

dBase-Datenbanken in Basic weiterverarbeiten

dBase besitzt zwar eine eigene Programmiersprache, doch ist die preiswerteste Programmiersprache immer noch GW-Basic – wird sie doch in der Regel bei neuen Computern zusammen mit dem Betriebssystem ausgeliefert. Und da die Mehrzahl der Daten im dBase-Format vorliegt, bietet sich ein Konvertierungsprogramm an, das dBase-Datenbanken auch in Basic verarbeiten läßt.

Sie sind Basic-Programmierer und schauen neidvoll auf den unverarbeiteten Datenbankbestand von Kollegen oder Bekannten? Mit der Konvertierungsroutine »dbasekon.bas« (Listing) machen Sie sich dBase-Datenbanken für Ihr GW-Basic verfügbar.

Nach Programmaufruf erscheint die Aufforderung, eine Diskette mit den dBase-Dateien in das aktuelle Laufwerk einzulegen. Die konvertierte Datei wird in das in Zeile 140 angegebene Laufwerk (»lw\$«) geschrieben. Nach Tastendruck werden alle dBase-Dateien, die die Erweiterung »dbf« haben, auf dem Bildschirm ausgegeben. Mit den Cursor-Tasten wird die Datei angewählt. [Enter] bestätigt die Eingabe. Stehen keine dBase-Dateien auf Diskette, erscheint eine Fehlermeldung, und das Programm startet nach Tastendruck erneut.

Es wird die Struktur der ausgesuchten dBase-Datei auf dem Bildschirm (Bild 1) ausgegeben, wobei jeweils zehn Feldnamen pro Seite angezeigt werden. Am Ende der Auflistung wird die Gesamtlänge eines Datensatzes ausgegeben. Danach entscheiden Sie, ob Sie die Daten in eine »sequentielle« oder in eine »wählfreie« Datei schreiben wollen.

Nach der Auswahl beginnt die Konvertierung. Es wird die »Laderoutine« nach untenstehenden Kriterien erzeugt und auf dem Bildschirm angezeigt. Zusätzlich erfolgt eine Warnung, falls die Satzlänge 128 Zeichen überschreitet. Durch Tastendruck kehrt das Programm auf die Betriebssystemebene zurück. Die generierte Datei und die Laderoutine befinden sich auf dem Laufwerk.

Bei der sequentiellen Laderoutine werden beim Ablauf alle Datenfelder in eine Tabelle geschrieben. Die Struktur der dBase-Datei sei die in Bild 2.

Sequentielle Laderoutine in Basic (Bild 3):

```
10 dim satz$(1,3)
```

wobei die erste Dimension für die Anzahl der gelesenen Datensätze (ein Datensatz) steht und die zweite Dimension für die Anzahl der Felder eines Datensatzes (drei Felder).

Sollen Datensätze hinzugefügt werden, so muß die erste Dimension erhöht werden.

```
20 open "I", #1, "example.dat"
```

Die Datei wird geöffnet – sie hat den gleichen Namen wie die dBase-Datei, jedoch mit der Endung ».dat« statt ».dbf«. Die Datendatei muß sich im aktuellen Laufwerk befinden.

```
30 satzsaehler=0
```

```
40 if eof(1) then close #1:goto 80
```

```
50 for x=1 to 3:input #1,satz$(satzsaehler+1,x):next
```

Die Datensätze der generierten Datei werden in die Tabelle übertragen.

```
60 satzsaehler=satzsaehler+1
```

```
70 goto 40
```

```
80 rem Größe der Tabelle steht in der Variablen satzsaehler!
```

Nach Ablauf dieser Routine stehen alle Daten in der Arbeitsspeichertabelle, wodurch sie schnell weiterverarbeitet werden können. Das erste Datum beginnt im Feld satz\$(1,1). Bei der sequentiellen Verarbeitung liegen die Grenzen nicht beim Programm, sondern bei der Programmiersprache, in diesem Falle Basic. Sollten die Dateien zu groß sein, um in Basic dimensioniert und geladen zu werden, müssen Sie in das wählfreie Format konvertieren.

Probleme bei der Laderoutine: Alle Feldnamen müssen in eine Zeile passen

Bei der wählfreien Laderoutine gibt es eventuell Probleme mit den Feldnamen, weil in Basic eine Programmzeile die Länge von 255 Zeichen nicht überschreiten darf und sich die Feldbeschreibung »field« nicht auf zwei Programmzeilen erstrecken kann.

Dieses Problem wird von dem Programm in drei Schritten gelöst:

1. Wenn die generierte Programmzeile nicht mehr als 255 Zeichen beträgt, werden die Originalnamen aus der dBase-Datei verwendet, zum Beispiel bei der Struktur der dBase-Datei, wie sie auch oben bei der sequentiellen Routine Verwendung finden. Die generierte Basic-Laderoutine würde dann lauten:

```
10 open "R", #1, "example1.dat", 45
```

45 ist die Gesamtlänge

```
20 field #1,20 as titel$,20 as autor$,5 as seriennr$
```

2. Wenn die generierte Programmzeile mit Originalnamen zwar 255 Zeichen überschreitet, aber jedes Feld noch einzeln beschrieben werden kann, wird jeder Feldname auf zwei Buchstaben, die das Programm vergibt, gekürzt

Bild 2.
Die Struktur einer
dBase-Datei

Feldname:	Typ:	Länge:	Dez.:
Titel	Zeichen	20	
Autor	Zeichen	20	
Seriennr.	Numerisch	5	0

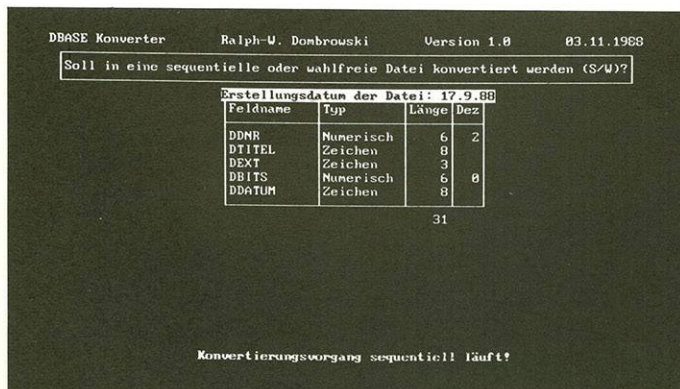


Bild 1. Die Datenbank zeigt ihre Felder und Feldlängen

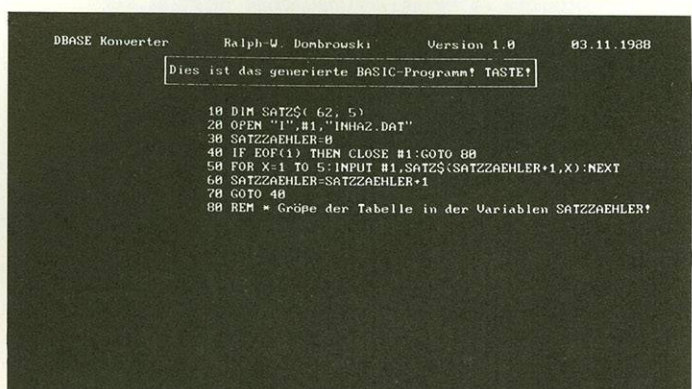


Bild 3. Die generierte Ladeprozedur für eine sequentielle Datei

zum Beispiel bei folgender Struktur einer dBase-Datei:

```
feldname10 Zeichen 5
feldname11 Zeichen 5
```

...

```
feldname30 Zeichen 5
```

Eine Programmzeile mit Originalnamen nimmt dann eine Länge von 368 Zeichen $(11 + (6 + 10 + 1) \times 21)$ an:

11 ergibt sich aus "20 field #1"

6 ergibt sich aus ",5 as"

10 ergibt sich aus der Länge der Feldnamen.

1 ergibt sich aus dem angehängten String-Zeichen "\$"

21 ergibt sich aus der Anzahl der Feldnamen.

Die generierte Basic-Laderoutine lautet dann folgendermaßen:

```
10 open "R", #1, "example2.dat", 121
```

```
20 field #1, 5 as aa$, 5 as ab$, ... , 5 as au$, 5 as av$
```

Dahinter werden die vergebenen Namen den Originalnamen zugeordnet:

```
30 rem aa$ = feldname 10
```

```
40 rem ab$ = feldname 11
```

...

```
240 rem av$ = feldname 30
```

3. Wenn auch dies noch nicht ausreicht, dann müssen Feldnamen zusammengelegt werden, bis die Maximalgröße erreicht ist, und später durch die Funktion »mid\$« wieder getrennt werden. Beispiel einer solchen Struktur bei einer dBase-Datei:

```
feldname10 Zeichen 10
```

```
feldname11 Zeichen 10
```

...

```
feldname40 Zeichen 10
```

Hier muß eine Programmzeile in Basic mit vom Programm vergebenen Namen (Beispiel 2) also 290 Zeichen $(11 + 9 \times 31)$ aufnehmen.

Die generierte Basic-Laderoutine lautet:

```
10 open "R", #1, "example3.dat", 310
```

```
20 field #1, 250 as a$, 60 as b$
```

Dahinter werden die zusammengelegten Felder ihren Originalnamen zugeordnet:

```
30 feldname10$ = mid$(a$, 1, 10)
```

```
40 feldname11$ = mid$(a$, 11, 10)
```

...

```
340 feldname40$ = mid$(b$, 51, 10)
```

Mit dieser Vorgehensweise werden auch die längsten Datensätze (die in dBase III maximal 4000 Byte betragen) erfaßt. Bei diesem Beispiel wird eine Warnung in bezug auf die Satzlänge von 310 $(250 + 60)$, siehe oben) ausgegeben werden.

dBase erlaubt in seinen Feldnamen übrigens den Unterstrich »_«; dieser ist in Basic nicht erlaubt, das heißt die generierte Laderoutine ist daraufhin zu überprüfen. So muß zum Beispiel der in dBase verwendete Feldname »serien_nr« für Basic umgewandelt werden in »seriennr\$«. Da dies jedoch zum Verlust der Eindeutigkeit eines Variablennamens führen kann, wird diese Umwandlung durch das Programm automatisiert.

Arbeiten Sie mit virtuellem Laufwerk und angepaßten Satzlängen, aber ohne Memo-Felder

Der voreingestellte Wert sieht für die Pufferlänge und damit die logische Satzlänge für Dateien nur 128 Byte vor. dBase-Dateien können aber bis zu 4000 Byte lang sein. Deshalb gibt das Programm eine Warnung aus, wie Basic in Zukunft zu starten ist, damit eine längere Satzlänge verarbeitet werden kann:

```
GWBasic /S:Satzlänge /I
```

Der Kommandointerpreter »command.com« muß erreichbar sein, weil das Programm mit dem Shell-Befehl arbeitet. Das Programm liefert keine Ergebnisse bei dBase-Dateien, die in der Struktur Memo-Felder aufweisen, da dBase in diesem Fall zwei Dateien anlegt. Weil das Programm

Fehler, wie zum Beispiel »Diskette voll«, nicht abfängt, kann es zum Abbruch kommen. An den beteiligten dBase-Dateien finden dadurch keine Veränderungen statt.

Das Programm läuft erheblich schneller, wenn es zu einer Exe-Datei kompiliert wird. Eine höhere Arbeitsgeschwindigkeit wird auch durch ein virtuelles Laufwerk erreicht, in dem sich »command.com«

```
set comspec=Laufwerksbuchstabe:\command.com
```

und die Dateien befinden.

(Ralph-Werner Dombrowski/al)

Zeile 130: date\$(128) – Dimensionierung ist willkürlich festgelegt. Auf der Auswahlbildschirmseite haben jedoch nicht mehr als 80 Dbf-Dateien Platz, sonst wird die Bildschirmmaske zerstört. Dies müßte in den meisten Fällen jedoch reichen.

feld\$(128) – maximale Anzahl von Feldern, die in dBase III/III Plus erlaubt sind: typ\$(128) lang\$(128) dez\$(128).

programm\$(130) – maximale Anzahl generierter Basic-Zeilen.

Zeile 190 bis 210: Das Diskettenverzeichnis wird in die Datei »dir.dat« umgeleitet und dann in die Tabelle datei\$ geladen.

Zeile 220 bis 240: Die Tabelle datei\$ wird umstrukturiert: Nur die Tabellenelemente mit der Erweiterung »dbf« werden berücksichtigt. Die Anzahl der auf Diskette stehenden dBase-Dateien ergibt sich aus y-1 und darf 80 nicht überschreiten, damit die Bildschirmmaske nicht zerstört wird.

Zeile 250 bis 280: Die Dateinamen werden zur Auswahl auf dem Bildschirm angezeigt.

Zeile 330 bis 400: Mittels Cursor- und Enter-Taste wird die gewünschte Datei ausgesucht.

Zeile 440 und 450: Der ausgesuchte Dateiname wird in d\$ gespeichert, und die Datei »d\$ + ".dbf"« wahlfrei mit einer Feldgröße von eins eröffnet.

Zeile 490 und 500: In einer Schleife wird festgestellt, wo der die Datei beschreibende Teil aufhört und der eigentliche Datenteil beginnt. Diese Stelle wird in »danfang« gespeichert.

Zeile 520: Das Entstehungsdatum der dBase-Datei ergibt sich aus dem zweiten, dritten und vierten Zeichen der Datei.

Zeile 540 und 550: Der Feldname hat eine Länge von 10 Byte, die Länge des Feldtyps beträgt 1 Byte, die Feldlänge 1 Byte, und falls typ\$ numerisch ist, enthält es 1 Byte für die Dezimalstellen.

Zeile 560 und 570: Die Feldnamen werden auf die sichtbaren ASCII-Zeichen beschränkt.

Zeile 600 bis 710: Die so ermittelte Dateistruktur wird auf dem Bildschirm ausgegeben (maximal 10 Feldnamen pro Seite) und die Gesamtlänge errechnet (lang\$).

Zeile 720 bis 750: Abfrage, ob eine sequentielle oder wahlfreie Datei erstellt werden soll. Anschließend die entsprechende Verzweigung.

Zeile 790: Die Zieldatei mit der Erweiterung »dat« wird eröffnet.

Zeile 800 bis 820: Die dBase-Datei wird Zeichen für Zeichen ab der Stelle »danfang« bis »eof« gelesen. In der Länge der Felder wird ein String aufgebaut und mittels Print-Befehl in die Zieldatei geschrieben.

Zeile 870: Die Zieldatei mit der Erweiterung ».dat« wird eröffnet.

Zeile 880/890: Zeichen für Zeichen wird in die Zieldatei bis zum eof (End-Of-File, auf deutsch »Dateiende«) übertragen.

Zeile 940: Warnung an den Anwender, falls Basic demnächst mit den entsprechenden Schaltern geladen werden muß.

Zeile 950 bis 980: Es wird die Basic-Laderoutine für sequentiellen Zugriff erzeugt und auf dem Bildschirm sowie auf das in Zeile 140 angegebene Laufwerk ausgegeben. Verzweigung zu den Endarbeiten

(Zeile 1190).

Zeile 1020 bis 1040: 1. Art der Field-Anweisung (Beispiel 1)

Zeile 1050: 2. Art der Field-Anweisung (Beispiel 2)

Zeile 1090 bis 1120: 3. Art der Field-Anweisung (Beispiel 3)

Zeile 1130: Ausgabe der Warnung, falls die Satzlänge größer als 128 ist.

Zeile 1140: Ausgabe der Open- und Field-Anweisung auf dem Bildschirm.

Zeile 1150: Das generierte Programm wird auf das in Zeile 140 angegebene Laufwerk geschrieben.

Zeile 1190 bis 1230: Ende des Programms

Zeile 1270 bis 1360: Unterprogramm für die Satzlängen-Warnung

Zeile 1400 bis 1420: Unterprogramm für den Hinweiskasten

Tabelle. Erläuterungen zum Listing »dbasekon.bas«


```

[NN4] 1 REM *****
[4F5] 2 REM Programm: »dbasekon.bas«
[Y55] 3 REM Prog.-Sprache: GWBasic 3.22
[P75] 4 REM Autor: Ralph-W. Dombrowski
[Q27] 5 REM (c) Redaktionsbüro Everts & Hagedorn
[2E2] 6 REM 1989
[6R6] 7 REM Konvertiert dBase-Dateien in Basic-
[422] 8 REM Dateien
[BM4] 100 REM *****
[7G5] 110 REM * Anfangsinitialisierung *
[9U3] 120 REM *****
[V8A] 130 DIM DATEIS(150), FIELDS(128), TYP$(128), LANG(128),
DEZ(128), PROGRAMM$(130)
[7WU] 140 COLOR 7,0:KEY OFF:CLS:PRINT"DBASE Konverter
Ralph-W. Dombrowski Version 1.0
03.11.1988";VIEW PRINT 2 TO 24:LEERS=STRING$(80,
32):STRICH$=STRING$(80,196):LW$="A:"
[2SC] 150 AS="Bitte legen Sie die Diskette mit den DBAS
E-Dateien ein! TASTE!":GOSUB 1400
[XM4] 160 REM *****
[FV6] 170 REM * Directory einladen und anzeigen*
[3N4] 180 REM *****
[UKA] 190 SHELL "dir">LW$+"dir.dat":DATEIZAEHLER=0:OPE
N "I",#1,LW$+"DIR.DAT"
[435] 200 IF EOF(1) THEN CLOSE #1:GOTO 220
[Y6B] 210 INPUT #1,DATEIS(DATEIZAEHLER):DATEIZAEHLER=DA
TEIZAEHLER+1:GOTO 200
[F15] 220 Y=1:FOR X=4 TO DATEIZAEHLER
[SCB] 230 IF MID$(DATEIS(X),1,0,3)="DBF" THEN DATEIS(Y)=
LEFT$(DATEIS(X),12)+" "":Y=Y+1
[LHL] 240 NEXT:DATEIZAEHLER=Y-1:IF DATEIZAEHLER=0 THEN
AS="Keine DBASE-Dateien auf Diskette! Taste!":VIE
W PRINT:GOSUB 1400:RUN
[MU5] 250 Z=1:X=5:WHILE X<=24:Y=1:WHILE Y<=4
[UOC] 260 LOCATE X,1+(Y-1)*20:PRINT DATEIS(Z):Z=Z+1:IF
Z>DATEIZAEHLER THEN X=X+1:Y=1
[WZ3] 270 Y=Y+1:WEND:X=X+1:WEND
[TH2] 280 LOCATE 5,1:COLOR 0,7:PRINT DATEIS(1);:SPOS=1:
ZPOS=1:SPALTANZ=4:ZEILANZ=INT(DATEIZAEHLER/4):IF
DATEIZAEHLER/4<INT(DATEIZAEHLER/4) THEN ZEILANZ=
ZEILANZ+1:SPALTANZ=(DATEIZAEHLER/4-INT(DATEIZAEHL
ER/4))*4:COLOR 7,0
[4L3] 290 REM *****
[3U4] 300 REM * Auswahl einer Datei *
[EK3] 310 REM *****
[92B] 320 AS="Bitte wählen Sie die zu konvertierende Da
tei aus!":GOSUB 1400
[UUV] 330 IF AS=CHR$(0)+CHR$(80) THEN LOCATE ZPOS+4,1+(
SPOS-1)*20:COLOR 7,0:PRINT DATEIS((ZPOS-1)*4+SPOS
):ZPOS=ZPOS+1:IF ZPOS>ZEILANZ OR (ZPOS=ZEILANZ A
ND SPOS>SPALTANZ) THEN ZPOS=1
[HKV] 340 IF AS=CHR$(0)+CHR$(72) THEN LOCATE ZPOS+4,1+(
SPOS-1)*20:COLOR 7,0:PRINT DATEIS((ZPOS-1)*4+SPOS
):ZPOS=ZPOS+1:IF ZPOS=0 THEN ZPOS=ZEILANZ:IF SPO
S>SPALTANZ THEN ZPOS=ZPOS-1
[UKW] 350 IF AS=CHR$(0)+CHR$(77) THEN LOCATE ZPOS+4,1+(
SPOS-1)*20:COLOR 7,0:PRINT DATEIS((ZPOS-1)*4+SPOS
):SPOS=SPOS+1:IF SPOS>4 THEN SPOS=1 ELSE IF ZPOS=
ZEILANZ AND SPOS>SPALTANZ THEN SPOS=1
[WXG] 360 IF AS=CHR$(0)+CHR$(75) THEN LOCATE ZPOS+4,1+(
SPOS-1)*20:COLOR 7,0:PRINT DATEIS((ZPOS-1)*4+SPOS
):SPOS=SPOS-1:IF SPOS=0 THEN SPOS=4:IF ZPOS=ZEILA
NZ AND SPOS>SPALTANZ THEN SPOS=SPALTANZ
[676] 370 IF AS=CHR$(13) THEN COLOR 7,0:GOTO 410
[RZA] 380 COLOR 0,7:LOCATE ZPOS+4,1+(SPOS-1)*20:PRINT D
ATEIS((ZPOS-1)*4+SPOS);
[W94] 390 AS=INKEY$:IF AS="" THEN 390
[7P1] 400 GOTO 330
[PG4] 410 REM *****
[BX6] 420 REM * Eröffnen der ausgewählten Datei
[WG4] 430 REM *****
[2F9] 440 X=INSTR(DATEIS((ZPOS-1)*4+SPOS),CHR$(32))-1:I
F X=0 THEN X=8
[B8B] 450 DS=LEFT$(DATEIS((ZPOS-1)*4+SPOS),X):OPEN "R",
#1,DS+"DBF",1:FIELD #1,1 AS Z$
[3N4] 460 REM *****
[DC7] 470 REM Struktur erstell./Datenanf. festst.
[9N4] 480 REM *****
[EKK] 490 X=32:ANFANG=1:WHILE ANFANG=1:GET #1,X+1:IF EO
F(1) THEN ANFANG=0 ELSE Z=ASC(Z$):IF Z=13 THEN AN
FANG=0:X=X+2 ELSE X=X+32
[K43] 500 WEND:DANFANG=X+1
[FE7] 510 REM Erstellungsdatum der Datei ermitteln
[RAX] 520 GET #1,4:DATUMS=RIGHT$(STR$(ASC(Z$)),LEN(STR$(
ASC(Z$)))-1)+".":GET #1,3:DATUMS=DATUMS+RIGHT$(S
TR$(ASC(Z$)),LEN(STR$(ASC(Z$)))-1)+".":GET #1,2:D
ATUMS=DATUMS+RIGHT$(STR$(ASC(Z$)),LEN(STR$(ASC(Z$
)))-1)
[686] 530 REM / Struktur auslese
n
[YEX] 540 FELD=1:Y=33:WHILE Y<X-32:FOR Z=0 TO 10:GET #1
,Y+Z:FELDS(FELD)=FELDS(FELD)+Z$:NEXT:GET #1,Y+Z:T
YP$(FELD)=Z$:GET #1,Y+Z+5:LANG(FELD)=ASC(Z$):IF T
YP$(FELD)="N" THEN GET #1,Y+Z+6:DEZ(FELD)=ASC(Z$)
[AS6] 550 FELD=FELD+1:Y=Y+32:WEND:FELD=FELD-1

```

```

[TRB] 560 FOR X=1 TO FELD:Y=INSTR(FELDS(X),CHR$(0)):IF
Y=0 THEN Y=LEN(FELDS(X))+1
[VG5] 570 FELDS(X)=LEFT$(FELDS(X),Y-1):NEXT
[DH4] 580 REM *****
[RS6] 590 REM * Struktur auf Bildschirm ausgeben
[RG4] 600 REM *****
[9FF] 610 CLS:Y=1:FOR X=1 TO FELD:IF Y=1 THEN COLOR 0,7
:LOCATE 5,23:PRINT"Erstellungsdatum der Datei: ";
DATUMS;:COLOR 7,0:LOCATE 6,23:PRINT" | Feldname |
Typ | Länge | Dez |":LOCATE 7,23:PRINT" |-----|
|-----|";
[S0A] 620 LOCATE 7+Y,23:PRINT " | FELDS(X);LEFT$(LEERS,1
1-LEN(FELDS(X)));";
[AP6] 630 IF TYP$(X)="C" THEN PRINT" | Zeichen |";
[QZ6] 640 IF TYP$(X)="N" THEN PRINT" | Numerisch |";
[UR6] 650 IF TYP$(X)="L" THEN PRINT" | Logisch |";
[FG6] 660 IF TYP$(X)="M" THEN PRINT" | Memo |";
[MK6] 670 IF TYP$(X)="D" THEN PRINT" | Datum |";
[WFG] 680 PRINT USING " |#####"LANG(X);:IF TYP$(X)="N"
THEN PRINT USING " |###"DEZ(X) ELSE PRINT" |";
[ACZ] 690 IF Y=10 AND X<FELD THEN LOCATE Y+8,23:PRINT"L
etzung TASTE!":GOSUB 1400:Y=0:CLS
[F82] 700 Y=Y+1:NEXT
[X41] 710 LANG=0:LOCATE Y+7,23:PRINT" |-----|";:FOR X=1 TO FELD:LANG=LANG+LANG
(X):NEXT:LOCATE Y+8,47:PRINT USING "#####"LANG;
[MME] 720 AS="Soll in eine sequentielle oder wahlfreie
Datei konvertiert werden (S/W)?":GOSUB 1400
[854] 730 IF AS="S" OR AS="s" THEN 790
[P54] 740 IF AS="W" OR AS="w" THEN 870
[9F2] 750 BEEP:GOTO 720
[RE4] 760 REM *****
[6D6] 770 REM * Sequentielle Datei erstellen *
[XE4] 780 REM *****
[XSM] 790 OPEN "o",#2,LW$+DS+"DAT":SATZZAEHLER=0:A=1:L
OCATE 24,20:COLOR 26,0:PRINT"Konvertierungsvorgan
g sequentiell läuft!":COLOR 7,0
[Q8B] 800 FOR X=1 TO FELD:AS="":FOR Y=1 TO LANG(X):GET
#1,DANFANG+A:AS=AS+Z$:A=A+1
[XM8] 810 NEXT:PRINT #2,AS:SATZZAEHLER=SATZZAEHLER+1:NE
XT
[XPF] 820 GET #1,DANFANG+A+1:IF EOF(1) THEN CLOSE:SATZZ
AEHLER=SATZZAEHLER/FELD:GOTO 830 ELSE A=A+1:GOTO
800
[DF5] 830 LOCATE 24,1:PRINT LEERS;:GOTO 940
[344] 840 REM *****
[BY5] 850 REM * Wahlfreie Datei erstellen *
[944] 860 REM *****
[W0N] 870 OPEN "r",#2,LW$+DS+"DAT",1:FIELD #2,1 AS U$:
A=1:Z=0:LOCATE 24,21:COLOR 26,0:PRINT"Konvertieru
ngsvorgang wahlfrei läuft!":COLOR 7,0
[H9E] 880 FOR X=1 TO FELD:FOR Y=1 TO LANG(X):GET #1,DAN
FANG+A:LSET U$=Z$:PUT #2,A-Z:A=A+1:NEXT:NEXT
[AMF] 890 GET #1,DANFANG+A+1:IF EOF(1) THEN CLOSE:SATZZ
AEHLER=(A-Z-1)/LANG:GOTO 900 ELSE A=A+1:Z=Z+1:GOT
O 880
[PK5] 900 LOCATE 24,1:PRINT LEERS;:GOTO 1020
[CE4] 910 REM *****
[9R6] 920 REM sequent. Einladeroutine in BASIC
[KE4] 930 REM *****
[GF7] 940 CLS:IF LANG>128 THEN X=14:GOSUB 1270 ELSE X=5
[64A] 950 PROGRAMM$(1)=" 10 DIM SATZ$(+STR$(SATZZAEHL
E R)+","+STR$(FELD)+")"
[S2G] 960 PROGRAMM$(2)=" 20 OPEN "+CHR$(34)+"+I"+CHR$(34
)+"",#1,"+CHR$(34)+DS+"DAT"+CHR$(34):PROGRAMM$(3)
=" 30 SATZZAEHLER=0"
[1DM] 965 PROGRAMM$(4)=" 40 IF EOF(1) THEN CLOSE #1:GOT
O 80":PROGRAMM$(5)=" 50 FOR X=1 TO "+STR$(FELD)+":
INPUT #1,SATZ$(SATZZAEHLER+1,X):NEXT"
[49R] 970 PROGRAMM$(6)=" 60 SATZZAEHLER=SATZZAEHLER+1":
PROGRAMM$(7)=" 70 GOTO 40":PROGRAMM$(8)=" 80 REM
* Größe der Tabelle in der Variablen SATZZAEHLER!"
[YHK] 980 OPEN "O",#3,LW$+DS+"BAS":FOR Y=1 TO 8:LOCATE
X+Y,20:PRINT PROGRAMM$(Y);:PRINT #3,PROGRAMM$(Y)
:NEXT:CLOSE #3:GOTO 1190
[SH4] 990 REM *****
[ET6] 1000 REM wahlfreie Einladeroutine in BASIC
[XE4] 1010 REM *****
[NPE] 1020 CLS:FELDLANG=12:FOR Y=1 TO FELD:FELDLANG=FEL
DLANG+LEN(STR$(LANG(Y)))+LEN(FELDS(Y))+5:NEXT
[FWG] 1025 PROGRAMM$(1)=" 10 OPEN "+CHR$(34)+"+R"+CHR$(3
4)+"+",#1,"+CHR$(34)+DS+"DAT"+CHR$(34)+","+STR$(LA
NG):PROGRAMM$(2)=" 20 FIELD #1"
[GC4] 1030 IF FELDLANG>255 THEN 1050
[9EL] 1040 FOR X=1 TO FELD:PROGRAMM$(2)=PROGRAMM$(2)+",
"+RIGHT$(STR$(LANG(X)),LEN(STR$(LANG(X)))-1)+" as
"+FELDS(X)+"$":NEXT:P=2:GOTO 1130
[R0C] 1050 FELDLANG=12:FOR Y=1 TO FELD:FELDLANG=FELDLA
NG+LEN(STR$(LANG(Y)))+7:NEXT

```

Listing. »dbasekon.bas« konvertiert dBase-Dateien ins ASCII-Format


```

[4D4] 1060 IF FELDLANG>255 THEN 1090
[N22] 1070 Y=65: Z=65
[CB0] 1075 FOR X=1 TO FELD:PROGRAMM$(2)=PROGRAMM$(2)+",
"RIGHT$(STR$(LANG(X)),LEN(STR$(LANG(X))-1)+" as
"+CHR$(Y)+CHR$(Z)+"$":PROGRAMM$(X+2)=STR$(10*(X+
2))+" REM * "+CHR$(Y)+CHR$(Z)+"$ = "+FELDS(X):Z=Z
+1:IF Z>90 THEN Y=Y+1:Z=65
[794] 1080 NEXT:P=2:FELD:GOTO 1130
[963] 1090 X=1:Z=65:Y=0:P=3
[K67] 1100 ANFANG=1:HILFLANG=LANG(X):WHILE ANFANG=1
[73P] 1105 IF HILFLANG+LANG(X+1)>255 THEN ANFANG=0:PROG
RAMM$(P)=STR$(X+2)*10+" "+FELDS(X)+"$=mid$("+CH
R$(Z)+"$,"+STR$(Y+1)+" "+STR$(LANG(X))+"$":P=P+1:
GOTO 1110
[6CU] 1106 IF X=FELD THEN ANFANG=0 ELSE PROGRAMM$(P)=S
TR$((X+2)*10)+" "+FELDS(X)+"$=mid$("+CHR$(Z)+"$,"
+STR$(Y+1)+" "+STR$(LANG(X))+"$":P=P+1:Y=HILFLANG
:X=X+1:HILFLANG=HILFLANG+LANG(X)
[1XG] 1110 WEND:X=X+1:PROGRAMM$(2)=PROGRAMM$(2)+" "+STR
$(HILFLANG)+" as "+CHR$(Z)+"$":Z=Z+1:IF X=FELD TH
EN Y=0:GOTO 1100
[VWE] 1120 PROGRAMM$(P)=STR$((FELD+2)*10)+" "+FELDS(FEL
D)+"$=mid$("+CHR$(Z-1)+"$,"+STR$(Y+1)+" "+STR$(LA
NG(FELD))+"$")
[TE7] 1130 CLS:IF LANG>128 THEN X=14:GOSUB 1270 ELSE X=
5
[BK8] 1140 FOR Y=1 TO 2:LOCATE X+Y,1:PRINT PROGRAMM$(Y)
:NEXT
[9LC] 1150 OPEN "O",#1,LW$+D$+".BAS":FOR X=1 TO P:PRINT
#1,PROGRAMM$(X):NEXT:CLOSE:GOTO 1190
[EH3] 1160 REM *****
[9T4] 1170 REM * Ende des Programms *
[PH3] 1180 REM *****
[UBA] 1190 A$="Dies ist das generierte BASIC-Programm!
TASTE!":GOSUB 1400
[5MA] 1200 CLS:LOCATE 20,1:PRINT"Die Dateien finden Sie
auf Laufwerk ";LW$
[LTB] 1210 PRINT"Kopieren Sie sich die Dateien mit der
Endung .DAT - Datendatei"
[BK8] 1220 PRINT"
.BAS - BASIC-Programm"

```

```

[7S3] 1230 VIEW PRINT:SYSTEM
[FF4] 1240 REM *****
[TF6] 1250 REM * BASIC mit Satzlänge starten *
[RF4] 1260 REM *****
[Y4P] 1270 LOCATE 5,20:PRINT"
";
[VEF] 1280 LOCATE 6,20:PRINT" | >>>>>>> "":COLOR 26,0
:PRINT"A C H T U N G";:COLOR 7,0:PRINT" <<<<<<<
< ";
[FB7] 1290 LOCATE 7,20:PRINT" |
";
[LYE] 1300 LOCATE 8,20:PRINT" | Die Satzlänge beträgt: "
:PRINT USING "#####";LANG;:PRINT" Bytes! ";
[PA7] 1310 LOCATE 9,20:PRINT" |
";
[CWA] 1320 LOCATE 10,20:PRINT" | Sie müssen BASIC demn
ächst mit ";
[67L] 1330 LOCATE 11,20:PRINT" | "":COLOR 13,0:P
RINT"CWBASIC /S:"":PRINT USING "#####";LANG;:PRIN
T" /I";:COLOR 7,0:PRINT" | ";
[GM8] 1340 LOCATE 12,20:PRINT" | aufrufen!
";
[FVP] 1350 LOCATE 13,20:PRINT" |
";
[449] 1360 COLOR 13,0:LOCATE X,1:PRINT STRICH$;:COLOR 7
,0:RETURN
[0C4] 1370 REM *****
[506] 1380 REM * Hinweis-Kasten mit Abfrage *
[8C4] 1390 REM *****
[WKL] 1400 FOR Z=1 TO 3:LOCATE 1+Z,1:PRINT LEERS$;:NEXT:
Z=(80-LEN(A$))/2-2:LOCATE 2,Z:PRINT" r";LEFT$(STRI
CH$,LEN(A$));"J";LOCATE 3,Z
[VFB] 1405 PRINT" |";A$;" |":LOCATE 4,Z:PRINT" L";LEFT$(S
TRICH$,LEN(A$));"J";
[GD4] 1410 A$=INKEY$:IF A$="" THEN 1410
[CV1] 1420 RETURN

```

Listing. »dbasekon.bas« konvertiert dBase-Dateien ins ASCII-Format

16 Farben auf CGA mit Turbo Basic

Nachdem bereits in DOS International 4'88 und in der letzten Ausgabe des DOS Extra über den 16-Farben-Modus der CGA-Farbgrafikkarte berichtet wurde, folgt diesmal eine Turbo-Basic-Bibliothek mit neun Funktionen für diesen Modus.

Der 16-Farben-Modus der CGA-Farbgrafikkarte läßt Redaktion und DOS-Leser nicht zur Ruhe kommen. Im folgenden finden Sie deshalb eine Turbo-Basic-Bibliothek mit neun Funktionen, die diesen speziellen, undokumentierten Modus der CGA-Karte unterstützen. Die zeitkritischen Routinen sind als Assembler-Inline-Code ausgeführt, um eine hohe Verarbeitungsgeschwindigkeit zu erreichen.

Die Turbo-Basic-Bibliothek enthält folgende Funktionen:

Initialisieren des CGA-Spezialmodus

Diese Funktion ruft den 16-Farben-Modus der CGA-Farbgrafikkarte auf und benötigt keine weiteren Parameter. Die Syntax lautet:

```
call CgaInit
```

Löschen des Bildschirms

Diese Funktion löscht den Bildschirm im 16-Farben-Modus in der gewünschten Farbe »farbe%«. Der Wertebereich für »farbe%« reicht von 0 bis 15, und die Syntax lautet:

```
call CgaCls(farbe%)
```

Setzen der Randfarbe

Diese Funktion definiert die Farbe »farbe%« für den Bildschirmrand. Der Wertebereich für »farbe%« reicht wiederum von 0 bis 15, und die Syntax lautet:

```
call CgaRand(farbe%)
```

Ausgabe von Text

Diese Funktion gibt den Text »text« in der Farbe »farbe%« auf die durch »zeile%« und »spalte%« markierte Textkoordinate aus. Der Wertebereich für »Zeile%« reicht von 1 bis 12 und der Wertebereich für »Spalte%« von 1 bis 20. Die Syntax für diese Funktion lautet:

```
call CgaPrint("text",farbe%,Zeile%,Spalte%)
```

Setzen eines Bildpunktes

Diese Funktion setzt einen Bildpunkt in der Farbe »farbe%« auf die durch »x%« und »y%« definierte Grafikkoordinate. Die Syntax dazu lautet:

```
call CgaPixel(x%,y%,farbe%)
```

Setzen eines Bildpunktes im Xor-Modus

Diese Funktion setzt einen Bildpunkt in der Farbe »farbe%« auf die durch »x%« und »y%« definierte Grafikkoordinate. Dabei wird die neue Farbe des Bildpunktes mit dessen alter Farbe über die logische Xor-Funktion verknüpft. Die Syntax lautet:

```
call CgaXorPixel(x%,y%,farbe%)
```

Abfragen der Farbe eines Bildpunktes

Diese Funktion ermittelt die Farbe des Bildpunktes auf der durch »x%« und »y%« definierten Grafikkoordinate. Die Syntax dafür lautet:

```
call CgaPoint(x%,y%,farbe%)
```

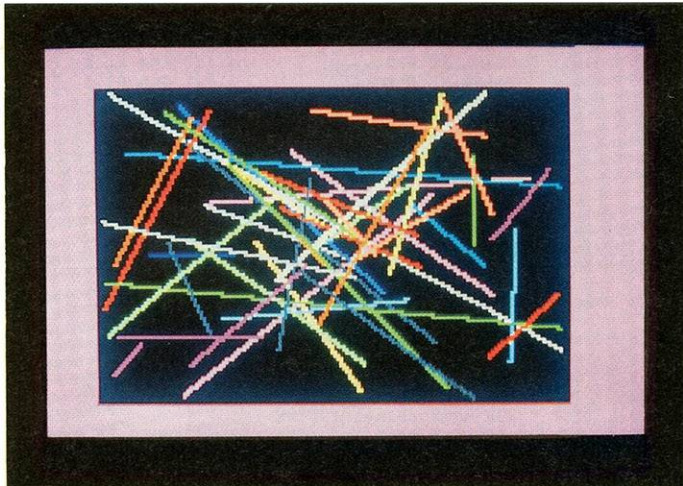



Bild 1. Ein farbenfrohes Beispiel aus der Vorführung von »demo.bas«

Zeichnen einer Linie

Diese Funktion zeichnet eine Linie in der Farbe »farbe%« zwischen den durch die Koordinaten »x1%,y1%« und »x2%,y2%« definierten Endpunkten. Die Syntax hierfür lautet:

```
call CgaLine(x1%,y1%,x2%,y2%,farbe%)
```

Zeichnen einer Linie im Xor-Modus

Diese Funktion zeichnet eine Linie in der Farbe »farbe%« zwischen den durch die Koordinaten »x1%,y1%« und »x2%,y2%« definierten Endpunkten. Dabei wird die Farbe

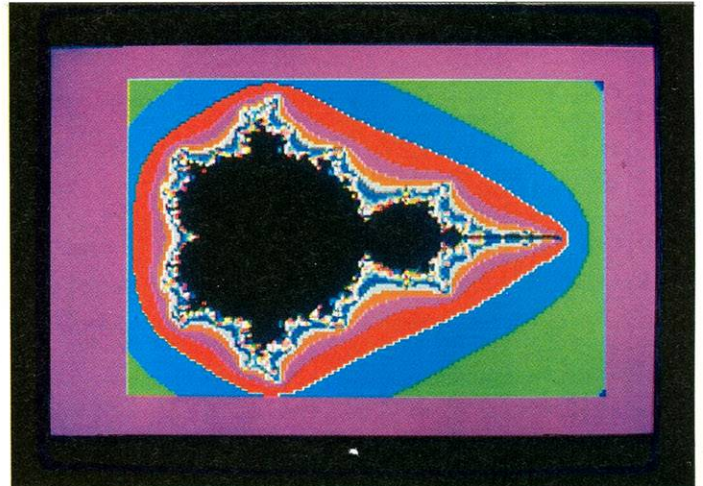


Bild 2. Das Ergebnis von »cgafrakt.bas« auf der CGA-Karte ist beeindruckend

der Linie mit der Hintergrundfarbe über die logische Xor-Funktion verknüpft. Die Syntax für diese Funktion lautet:

```
call CgaXorLine(x1%,y1%,x2%,y2%,farbe%)
```

Listing 2 und 3 zeigen zwei Anwendungen der Turbo-Basic-Bibliothek. Listing 2 führt die Fähigkeiten der einzelnen Funktionen aus der Bibliothek vor (Bild 1) und Listing 3 zeichnet das Apfelmännchen-Fraktal in 16 Farben (Bild 2). Die Programme funktionieren auch auf dem Amstrad PC 1512, obwohl dessen Grafik nicht voll CGA-kompatibel ist.

(Jürgen Heisig/ma)

```
[EN1] 1: "CGA.INC"
[000] 2:
[RFK] 3:
[A56] 4: Turbo-Basic - Subroutinen für
[9S5] 5: den CGA-Spezialmodus (160*100/16)
[RFK] 6:
[VTM] 7:

[4D6] 8: 'Initialisieren des CGA-Modus 160*100/16:
[Y6L] 9:
[AE3] 10: 'Aufruf: call CgaInit
[000] 11:
[W12] 12: sub CgaInit
[5N1] 13: local a%,b%
[CE1] 14: CrtData:
[MD4] 15: data 113,80,90,10,127,6,100,112,0,1
[BG1] 16: screen 0,1
[PN0] 17: cls
[TR1] 18: out &h3d8,1
[WU2] 19: restore CrtData
[002] 20: for a%=0 to 9
[N21] 21: read b%
[PU1] 22: out &h3d4,a%
[WU1] 23: out &h3d5,b%
[P51] 24: next a%
[472] 25: call CgaCls(0)
[HA1] 26: end sub
[7SM] 27:

[VTM] 28:

[XT3] 29: 'setzen der Randfarbe:
[27A] 30:
[WK4] 31: 'Aufruf: call CgaRand(farbe%)
[000] 32:
[S43] 33: sub CgaRand(farbe%)
[EX2] 34: farbe%=farbe% and 15
[LH2] 35: out &h3d9,farbe%
[HA1] 36: end sub
[7SM] 37:

[0BU] 38:
```

```
[A36] 39: 'Schriftausgabe im CGA-Modus 160*100/16:
[KMK] 40:
[2W6] 41: '(Begrenzung auf die Zeichen bis ASCII 12
7)
[MC8] 42: 'Aufruf: call CgaPrint("text",farbe%,Zeil
e%,Spalte%)
[5D7] 43: 'wobei Farbe%=0...15, Zeile%=1...12, Spal
te%=1...20

[000] 44:
[TT4] 45: sub CgaPrint(text$,ff%,yy%,xx%)
[LC4] 46: local posx%,posy%,j%,u%,z%,k%
[NE4] 47: local zweihochbit%,bit%,x%,y%
[TS2] 48: def seg=&hff0000
[227] 49: if xx%<1 or xx%>20 or yy%<1 or yy%>12
then exit sub
[FB2] 50: posy%=(yy%-1)*8+1
[BB2] 51: posx%=(xx%-1)*8+1
[UT3] 52: for j%=0 to len(text$)-1
[VM3] 53: u%=asc(mid$(text$,j%+1,1))
[902] 54: if u%<128 then
[C11] 55: u%=8*u%
[E12] 56: for z%=0 to 7
[A23] 57: k%=peek(&hfa6e+z%+u%)
[611] 58: jj%=j%*8
[YY0] 59: bit%=0
[8E2] 60: zweihochbit%=1
[9L3] 61: while zweihochbit%<129
[R64] 62: if k% and zweihochbit% then
[L03] 63: x%=posx%+jj%+7-bit%
[XV1] 64: y%=posy%+z%
[YS3] 65: call CgaPixel(x%,y%,ff%)
[4H2] 66: end if '(k% and...
[2D1] 67: incr bit%
[755] 68: incr zweihochbit%,zweihochbit%
[TR0] 69: wend
[101] 70: next z%
[8E1] 71: end if 'u%
[H61] 72: next j%
[091] 73: def seg
[HA1] 74: end sub
[NKT] 75:

[EN0] 76:
```

Listing 1. Die Include-Datei »cga.inc« enthält die Funktionen für den 16-Farben-Modus der CGA-Karte


```
[RD6] 77: 'Löschen des Bildschirms, Farbe (0-15):
[KKM] 78:
[ZB6] 79: 'Bildschirm wird dabei aus- und danach
[TP3] 80: 'wieder eingeschaltet
[UC4] 81: 'Aufruf: call CgaCIs(farbe%)
[000] 82:
[5U2] 83: SUB CgaCIs INLINE
[3K4] 84: $INLINE &h55 '0100 PUSH BP
[CK4] 85: $INLINE &h1E '0101 PUSH DS
[PC5] 86: $INLINE &h8B,&hEC '0102 MOV BP,SP
[S56] 87: $INLINE &hC5,&h7E,&h08 '0104 LDS DI,[BP
+08]
[DL5] 88: $INLINE &h8A,&h3D '0107 MOV BH,[DI
]
[VB5] 89: $INLINE &h80,&hE7,&h0F '0109 AND BH,0F
[MA5] 90: $INLINE &h8A,&hDF '010C MOV BL,BH
[U24] 91: $INLINE &hD0,&hE3 '010E SHL BL,1
[924] 92: $INLINE &hD0,&hE3 '0110 SHL BL,1
[B24] 93: $INLINE &hD0,&hE3 '0112 SHL BL,1
[D24] 94: $INLINE &hD0,&hE3 '0114 SHL BL,1
[C25] 95: $INLINE &h0A,&hFB '0116 OR BH,BL
[ZA5] 96: $INLINE &hB3,&hDE '0118 MOV BL,DE
[XL5] 97: $INLINE &hB8,&h00,&hB8 '011A MOV AX,B80
0
[0C5] 98: $INLINE &h8E,&hC0 '011D MOV ES,AX
[EB5] 99: $INLINE &h33,&hFF '011F XOR DI,DI
[FM5] 100: $INLINE &hB9,&h40,&h1F '0121 MOV CX,1F4
0
[C04] 101: $INLINE &hFC '0124 CLD
[NK5] 102: $INLINE &hBA,&hD8,&h03 '0125 MOV DX,03D
8
[M45] 103: $INLINE &hB0,&h01 '0128 MOV AL,01
[345] 104: $INLINE &hEE '012A OUT DX,AL
[XB5] 105: $INLINE &h8B,&hC3 '012B MOV AX,BX
[7B4] 106: $INLINE &hF3 '012D REPZ
[VN4] 107: $INLINE &hAB '012E STOSW
[6L5] 108: $INLINE &hBA,&hD8,&h03 '012F MOV DX,03D
8
[265] 109: $INLINE &hB0,&h09 '0132 MOV AL,09
[U35] 110: $INLINE &hEE '0134 OUT DX,AL
[KE4] 111: $INLINE &h1F '0135 POP DS
[PE4] 112: $INLINE &h5D '0136 POP BP
[HA1] 113: END SUB
[MM0] 114:
[000] 115:
[EN0] 116:
[CW7] 117: 'Setzen eines Bildpunktes im CGA-Modus 16
0*100/16:
[Y6L] 118:
[ZL5] 119: 'Aufruf: call CgaPixel(x%,y%,farbe%)
[000] 120:
[5D3] 121: SUB CgaPixel INLINE
[3K4] 122: $INLINE &h55 '0100 PUSH BP
[CK4] 123: $INLINE &h1E '0101 PUSH DS
[PC5] 124: $INLINE &h8B,&hEC '0102 MOV BP,SP
[S56] 125: $INLINE &hC5,&h7E,&h08 '0104 LDS DI,[BP
+08]
[DL5] 126: $INLINE &h8A,&h3D '0107 MOV BH,[DI
]
[VB5] 127: $INLINE &h80,&hE7,&h0F '0109 AND BH,0F
[M86] 128: $INLINE &hC5,&h7E,&h0C '010C LDS DI,[BP
+0C]
[YL5] 129: $INLINE &h8A,&h1D '010F MOV BL,[DI
]
[646] 130: $INLINE &hC5,&h7E,&h10 '0111 LDS DI,[BP
+10]
[0M5] 131: $INLINE &h8B,&h0D '0114 MOV CX,[DI
]
[KL5] 132: $INLINE &hB8,&h00,&hB8 '0116 MOV AX,B80
0
[2C5] 133: $INLINE &h8E,&hD8 '0119 MOV DS,AX
[HB5] 134: $INLINE &h33,&hC0 '011B XOR AX,AX
[Y95] 135: $INLINE &h8A,&hC3 '011D MOV AL,BL
[205] 136: $INLINE &hD1,&hE0 '011F SHL AX,1
[H24] 137: $INLINE &hD1,&hE0 '0121 SHL AX,1
[L24] 138: $INLINE &hD1,&hE0 '0123 SHL AX,1
[N24] 139: $INLINE &hD1,&hE0 '0125 SHL AX,1
[R24] 140: $INLINE &hD1,&hE0 '0127 SHL AX,1
[VB5] 141: $INLINE &h8B,&hF8 '0129 MOV DI,AX
[205] 142: $INLINE &hD1,&hE0 '012B SHL AX,1
[405] 143: $INLINE &hD1,&hE0 '012D SHL AX,1
[U75] 144: $INLINE &h03,&hF8 '012F ADD DI,AX
[H75] 145: $INLINE &h03,&hF9 '0131 ADD DI,CX
[AG6] 146: $INLINE &hF7,&hC1,&h01,&h00 '0133 TEST
CX,0001
[9W4] 147: $INLINE &h75,&h01 '0137 JNZ 013A
[KC4] 148: $INLINE &h47 '0139 INC DI
[2L5] 149: $INLINE &h8A,&h05 '013A MOV AL,[DI
]
[AH6] 150: $INLINE &hF7,&hC1,&h01,&h00 '013C TEST
CX,0001
[5N4] 151: $INLINE &h74,&h06 '0140 JZ 0148
[R25] 152: $INLINE &h24,&hF0 '0142 AND AL,F0
[455] 153: $INLINE &h02,&hC7 '0144 ADD AL,BH
[XX4] 154: $INLINE &hEB,&h0C '0146 JMP 0154
```

```
[945] 155: $INLINE &h24,&h0F '0148 AND AL,0F
[A05] 156: $INLINE &hD0,&hE7 '014A SHL BH,1
[C05] 157: $INLINE &hD0,&hE7 '014C SHL BH,1
[E05] 158: $INLINE &hD0,&hE7 '014E SHL BH,1
[X24] 159: $INLINE &hD0,&hE7 '0150 SHL BH,1
[655] 160: $INLINE &h02,&hC7 '0152 ADD AL,BH
[AL5] 161: $INLINE &h88,&h05 '0154 MOV [DI],A
L
[VE4] 162: $INLINE &h1F '0156 POP DS
[ZEA] 163: $INLINE &h5D '0157 POP BP
[HA1] 164: END SUB
[MM0] 165:
[EN0] 166:
[A76] 167: 'Setzen eines Bildpunktes, Xor-Modus:
[60F] 168:
[AD6] 169: 'Aufruf: call CgaXorPixel(x%,y%,farbe%)
[000] 170:
[GY3] 171: SUB CgaXorPixel INLINE
[3K4] 172: $INLINE &h55 '0100 PUSH BP
[CK4] 173: $INLINE &h1E '0101 PUSH DS
[PC5] 174: $INLINE &h8B,&hEC '0102 MOV BP,SP
[S56] 175: $INLINE &hC5,&h7E,&h08 '0104 LDS DI,[BP
+08]
[DL5] 176: $INLINE &h8A,&h3D '0107 MOV BH,[DI
]
[VB5] 177: $INLINE &h80,&hE7,&h0F '0109 AND BH,0F
[M86] 178: $INLINE &hC5,&h7E,&h0C '010C LDS DI,[BP
+0C]
[YL5] 179: $INLINE &h8A,&h1D '010F MOV BL,[DI
]
[646] 180: $INLINE &hC5,&h7E,&h10 '0111 LDS DI,[BP
+10]
[0M5] 181: $INLINE &h8B,&h0D '0114 MOV CX,[DI
]
[KL5] 182: $INLINE &hB8,&h00,&hB8 '0116 MOV AX,B80
0
[2C5] 183: $INLINE &h8E,&hD8 '0119 MOV DS,AX
[HB5] 184: $INLINE &h33,&hC0 '011B XOR AX,AX
[Y95] 185: $INLINE &h8A,&hC3 '011D MOV AL,BL
[205] 186: $INLINE &hD1,&hE0 '011F SHL AX,1
[H24] 187: $INLINE &hD1,&hE0 '0121 SHL AX,1
[L24] 188: $INLINE &hD1,&hE0 '0123 SHL AX,1
[N24] 189: $INLINE &hD1,&hE0 '0125 SHL AX,1
[R24] 190: $INLINE &hD1,&hE0 '0127 SHL AX,1
[VB5] 191: $INLINE &h8B,&hF8 '0129 MOV DI,AX
[205] 192: $INLINE &hD1,&hE0 '012B SHL AX,1
[405] 193: $INLINE &hD1,&hE0 '012D SHL AX,1
[U75] 194: $INLINE &h03,&hF8 '012F ADD DI,AX
[H75] 195: $INLINE &h03,&hF9 '0131 ADD DI,CX
[AC6] 196: $INLINE &hF7,&hC1,&h01,&h00 '0133 TEST
CX,0001
[9W4] 197: $INLINE &h75,&h01 '0137 JNZ 013A
[KC4] 198: $INLINE &h47 '0139 INC DI
[2L5] 199: $INLINE &h8A,&h05 '013A MOV AL,[DI
]
[AH6] 200: $INLINE &hF7,&hC1,&h01,&h00 '013C TEST
CX,0001
[XM4] 201: $INLINE &h74,&h04 '0140 JZ 0146
[H95] 202: $INLINE &h32,&hC7 '0142 XOR AL,BH
[BX4] 203: $INLINE &hEB,&h0A '0144 JMP 0150
[Z24] 204: $INLINE &hD0,&hE7 '0146 SHL BH,1
[105] 205: $INLINE &hD0,&hE7 '0148 SHL BH,1
[A05] 206: $INLINE &hD0,&hE7 '014A SHL BH,1
[C05] 207: $INLINE &hD0,&hE7 '014C SHL BH,1
[4A5] 208: $INLINE &h32,&hC7 '014E XOR AL,BH
[6L5] 209: $INLINE &h88,&h05 '0150 MOV [DI],A
L
[RE4] 210: $INLINE &h1F '0152 POP DS
[VE4] 211: $INLINE &h5D '0153 POP BP
[HA1] 212: END SUB
[MM0] 213:
[EN0] 214:
[B86] 215: 'Abfragen der Farbe eines Bildpunktes:
[ECF] 216:
[KN5] 217: 'Aufruf: call CgaPoint(x%,y%,farbe%)
[000] 218:
[GD3] 219: SUB CgaPoint INLINE
[3K4] 220: $INLINE &h55 '0100 PUSH BP
[CK4] 221: $INLINE &h1E '0101 PUSH DS
[PC5] 222: $INLINE &h8B,&hEC '0102 MOV BP,SP
[S86] 223: $INLINE &hC5,&h7E,&h0C '0104 LDS DI,[BP
+0C]
[FL5] 224: $INLINE &h8A,&h1D '0107 MOV BL,[DI
]
[A46] 225: $INLINE &hC5,&h7E,&h10 '0109 LDS DI,[BP
+10]
[BM5] 226: $INLINE &h8B,&h0D '010C MOV CX,[DI
]
[XL5] 227: $INLINE &hB8,&h00,&hB8 '010E MOV AX,B80
0
[UB5] 228: $INLINE &h8E,&hD8 '0111 MOV DS,AX
[ZB5] 229: $INLINE &h33,&hC0 '0113 XOR AX,AX
[F95] 230: $INLINE &h8A,&hC3 '0115 MOV AL,BL
```



```

[LZ4] 231: $INLINE &hD1,&hE0 '0117 SHL AX,1
[NZ4] 232: $INLINE &hD1,&hE0 '0119 SHL AX,1
[YZ4] 233: $INLINE &hD1,&hE0 '011B SHL AX,1
[005] 234: $INLINE &hD1,&hE0 '011D SHL AX,1
[205] 235: $INLINE &hD1,&hE0 '011F SHL AX,1
[LB5] 236: $INLINE &h8B,&hF8 '0121 MOV DI,AX
[LZ4] 237: $INLINE &hD1,&hE0 '0123 SHL AX,1
[NZ4] 238: $INLINE &hD1,&hE0 '0125 SHL AX,1
[B75] 239: $INLINE &h03,&hF8 '0127 ADD DI,AX
[N75] 240: $INLINE &h03,&hF9 '0129 ADD DI,CX
[7H6] 241: $INLINE &hF7,&hC1,&h01,&h00 '012B TEST
CX,0001
[7V4] 242: $INLINE &h75,&h01 '012F JNZ 0132
[AC4] 243: $INLINE &h47 '0131 INC DI
[LK5] 244: $INLINE &h8A,&h05 '0132 MOV AL,[DI
]
[CG6] 245: $INLINE &hF7,&hC1,&h01,&h00 '0134 TEST
CX,0001
[2P4] 246: $INLINE &h74,&h05 '0138 JZ 013F
[E45] 247: $INLINE &h24,&h0F '013A AND AL,0F
[HY4] 248: $INLINE &hEB,&h0B '013C JMP 0149
[114] 249: $INLINE &h90 '013E NOP
[735] 250: $INLINE &h24,&h0F '013F AND AL,F0
[805] 251: $INLINE &hD0,&hE8 '0141 SHR AL,1
[A05] 252: $INLINE &hD0,&hE8 '0143 SHR AL,1
[C05] 253: $INLINE &hD0,&hE8 '0145 SHR AL,1
[E05] 254: $INLINE &hD0,&hE8 '0147 SHR AL,1
[D95] 255: $INLINE &h32,&hE4 '0149 XOR AH,AH
[P66] 256: $INLINE &hC5,&h7E,&h08 '014B LDS DI,[BP
+08]
[GM5] 257: $INLINE &h89,&h05 '014E MOV [DI],A
X
[NE4] 258: $INLINE &h1F '0150 POP DS
[TE4] 259: $INLINE &h5D '0151 POP BP
[HA1] 260: END SUB
[MM0] 261:
[EN0] 262:
[UD6] 263: 'Zeichnen einer Linie (x1,y1,x2,y2,farbe)
[Y6L] 264:
[001] 265: 'Aufruf:
[0L5] 266: 'call CgaLine(x1%,y1%,x2%,y2%,farbe%)
[000] 267:
[843] 268: SUB CgaLine INLINE
[FW4] 269: $INLINE &hEB,&h53 '0100 JMP 0155
[736] 270: $INLINE &h00,&h00 '0102 Argument 3
(x2%)
[E36] 271: $INLINE &h00,&h00 '0104 Argument 4
(y2%)
[UR4] 272: $INLINE &h00,&h00 '0106 DeltaX
[5T4] 273: $INLINE &h00,&h00 '0108 YStep
[DU4] 274: $INLINE &h00,&h00 '010A Old_x
[LU4] 275: $INLINE &h00,&h00 '010C Old_y
[TE6] 276: $INLINE &h00 '010E Argument 5
(farbe%)
[7L4] 277: $INLINE &h53 '010F PUSH BX
[XH4] 278: $INLINE &h57 '0110 PUSH DI
[7M6] 279: $INLINE &h8A,&h3E,&h0E,&h01 '0111 MOV B
H,[010E]
[4B5] 280: $INLINE &h33,&hC0 '0115 XOR AX,AX
[H95] 281: $INLINE &h8A,&hC3 '0117 MOV AL,BL
[NZ4] 282: $INLINE &hD1,&hE0 '0119 SHL AX,1
[YZ4] 283: $INLINE &hD1,&hE0 '011B SHL AX,1
[005] 284: $INLINE &hD1,&hE0 '011D SHL AX,1
[205] 285: $INLINE &hD1,&hE0 '011F SHL AX,1
[H24] 286: $INLINE &hD1,&hE0 '0121 SHL AX,1
[NB5] 287: $INLINE &h8B,&hF8 '0123 MOV DI,AX
[NZ4] 288: $INLINE &hD1,&hE0 '0125 SHL AX,1
[RZ4] 289: $INLINE &hD1,&hE0 '0127 SHL AX,1
[D75] 290: $INLINE &h03,&hF8 '0129 ADD DI,AX
[H85] 291: $INLINE &h03,&hFA '012B ADD DI,DX
[FH6] 292: $INLINE &hF7,&hC2,&h01,&h00 '012D TEST
DX,0001
[WU4] 293: $INLINE &h75,&h01 '0131 JNZ 0134
[CC4] 294: $INLINE &h47 '0133 INC DI
[9A6] 295: $INLINE &h26,&h8A,&h05 '0134 MOV AL,ES:
[DI]
[PG6] 296: $INLINE &hF7,&hC2,&h01,&h00 '0137 TEST
DX,0001
[9N4] 297: $INLINE &h74,&h06 '013B JZ 0143
[535] 298: $INLINE &h24,&h0F '013D AND AL,F0
[K55] 299: $INLINE &h02,&hC7 '013F ADD AL,BH
[C24] 300: $INLINE &hEB,&h0C '0141 JMP 014F
[445] 301: $INLINE &h24,&h0F '0143 AND AL,0F
[YZ4] 302: $INLINE &hD0,&hE7 '0145 SHL BH,1
[005] 303: $INLINE &hD0,&hE7 '0147 SHL BH,1
[205] 304: $INLINE &hD0,&hE7 '0149 SHL BH,1
[B05] 305: $INLINE &hD0,&hE7 '014B SHL BH,1
[M55] 306: $INLINE &h02,&hC7 '014D ADD AL,BH
[096] 307: $INLINE &h26,&h88,&h05 '014F MOV ES:[DI
],AL
[XE4] 308: $INLINE &h5F '0152 POP DI
[1F4] 309: $INLINE &h5B '0153 POP BX
[H24] 310: $INLINE &hC3 '0154 RET
[000] 311:

```

```

[WK4] 312: $INLINE &h55 '0155 PUSH BP
[5L4] 313: $INLINE &h1E '0156 PUSH DS
[FD5] 314: $INLINE &h8B,&hEC '0157 MOV BP,SP
[H66] 315: $INLINE &hC4,&h7E,&h08 '0159 LES DI,[BP
+08]
[LC6] 316: $INLINE &h26,&h8A,&h1D '015C MOV BL,ES:
[DI]
[SC5] 317: $INLINE &h80,&hE3,&h0F '015F AND BL,0F
[LG6] 318: $INLINE &h88,&h1E,&h0E,&h01 '0162 MOV [
010E],BL
[Z86] 319: $INLINE &hC4,&h7E,&h0C '0166 LES DI,[BP
+0C]
[VC6] 320: $INLINE &h26,&h8B,&h1D '0169 MOV BX,ES:
[DI]
[YF6] 321: $INLINE &h89,&h1E,&h04,&h01 '016C MOV [
0104],BX
[5D5] 322: $INLINE &h8B,&hCB '0170 MOV CX,BX
[Z46] 323: $INLINE &hC4,&h7E,&h10 '0172 LES DI,[BP
+10]
[VC6] 324: $INLINE &h26,&h8B,&h1D '0175 MOV BX,ES:
[DI]
[1F6] 325: $INLINE &h89,&h1E,&h02,&h01 '0178 MOV [
0102],BX
[EE5] 326: $INLINE &h8B,&hF3 '017C MOV SI,BX
[E66] 327: $INLINE &hC4,&h7E,&h14 '017E LES DI,[BP
+14]
[VC6] 328: $INLINE &h26,&h8B,&h1D '0181 MOV BX,ES:
[DI]
[XD5] 329: $INLINE &h85,&hDB '0184 TEST BX,BX
[8X4] 330: $INLINE &h79,&h04 '0186 JNS 018C
[DN6] 331: $INLINE &h8B,&h1E,&h0C,&h01 '0188 MOV B
X,[010C]
[C76] 332: $INLINE &hC4,&h7E,&h18 '018C LES DI,[BP
+18]
[BC6] 333: $INLINE &h26,&h8B,&h15 '018F MOV DX,ES:
[DI]
[9D5] 334: $INLINE &h85,&hD2 '0192 TEST DX,DX
[6X4] 335: $INLINE &h79,&h04 '0194 JNS 019A
[0M6] 336: $INLINE &h8B,&h16,&h0A,&h01 '0196 MOV D
X,[010A]
[L46] 337: $INLINE &hA1,&h02,&h01 '019A MOV AX,[01
02]
[086] 338: $INLINE &hA3,&h0A,&h01 '019D MOV [010A]
,AX
[C56] 339: $INLINE &hA1,&h04,&h01 '01A0 MOV AX,[01
04]
[V86] 340: $INLINE &hA3,&h0C,&h01 '01A3 MOV [010C]
,AX
[KN5] 341: $INLINE &hB8,&h00,&hB8 '01A6 MOV AX,B80
[ND5] 342: $INLINE &h8E,&hC0 '01A9 MOV ES,AX
[HD5] 343: $INLINE &h33,&hC0 '01AB XOR SI,AX
[TE5] 344: $INLINE &h2B,&hF2 '01AD SUB SI,DX
[VE5] 345: $INLINE &h1D,&h00,&h00 '01AF SBB AX,000
[KA5] 346: $INLINE &h03,&hF0 '01B2 ADD SI,AX
[3F5] 347: $INLINE &h33,&hF0 '01B4 XOR SI,AX
[5E5] 348: $INLINE &h0D,&h01,&h00 '01B6 OR AX,0001
[VD5] 349: $INLINE &h8B,&hF8 '01B9 MOV DI,AX
[9H6] 350: $INLINE &h89,&h36,&h06,&h01 '01BB MOV [
0106],SI
[8G4] 351: $INLINE &h4E '01BF DEC SI
[XP4] 352: $INLINE &h78,&h02 '01C0 JS 01C4
[BF5] 353: $INLINE &h33,&hF6 '01C2 XOR SI,SI
[BD5] 354: $INLINE &h33,&hC0 '01C4 XOR AX,AX
[TD5] 355: $INLINE &h2B,&hCB '01C6 SUB CX,BX
[NE5] 356: $INLINE &h1D,&h00,&h00 '01C8 SBB AX,000
[2A5] 357: $INLINE &h03,&hC8 '01CB ADD CX,AX
[LE5] 358: $INLINE &h33,&hC8 '01CD XOR CX,AX
[TE5] 359: $INLINE &h0D,&h01,&h00 '01CF OR AX,0001
[476] 360: $INLINE &hA3,&h08,&h01 '01D2 MOV [0108]
,AX
[YC6] 361: $INLINE &h39,&h16,&h02,&h01 '01D5 CMP [
0102],DX
[RY4] 362: $INLINE &h75,&h08 '01D9 JNZ 01E3
[PE6] 363: $INLINE &h39,&h1E,&h04,&h01 '01DB CMP [
0104],BX
[SY4] 364: $INLINE &h75,&h02 '01DF JNZ 01E3
[L15] 365: $INLINE &hEB,&h18 '01E1 JMP 01FB
[VG5] 366: $INLINE &hE8,&h29,&hFF '01E3 CALL 010F
[RE5] 367: $INLINE &h83,&hFE,&h00 '01E6 CMP SI,+00
[LY4] 368: $INLINE &h7D,&h0A '01E9 JGE 01F5
[UF6] 369: $INLINE &h03,&h1E,&h08,&h01 '01EB ADD B
X,[0108]
[BE6] 370: $INLINE &h03,&h36,&h06,&h01 '01EF ADD S
I,[0106]
[K05] 371: $INLINE &hEB,&hE0 '01F3 JMP 01D5
[H85] 372: $INLINE &h03,&hD7 '01F5 ADD DX,DI
[VE5] 373: $INLINE &h2B,&hF1 '01F7 SUB SI,CX
[T15] 374: $INLINE &hEB,&hDA '01F9 JMP 01D5
[BH4] 375: $INLINE &h1F '01FB POP DS
[FH4] 376: $INLINE &h5D '01FC POP BP
[HA1] 377: END SUB
[MM0] 378:

```

Listing 1. Die Include-Datei »cga.inc« enthält die Funktionen für den 16-Farben-Modus der CGA-Karte


```
[EN0] 379: '
[UD6] 380: 'Zeichnen einer Linie (x1,y1,x2,y2,farbe)
[192] 381: 'im XOR-Modus
[Y6L] 382: '
[0D1] 383: 'Aufruf:
[M36] 384: 'call CgaXorLine(x1%,y1%,x2%,y2%,farbe%)
[000] 385: '
[1K3] 386: SUB CgaXorLine INLINE
[8X4] 387: $INLINE &hEB,&h4F '0100 JMP 0151
[736] 388: $INLINE &h00,&h00 '0102 Argument 3
(x2%)
[E36] 389: $INLINE &h00,&h00 '0104 Argument 4
(y2%)
[UR4] 390: $INLINE &h00,&h00 '0106 DeltaX
[5T4] 391: $INLINE &h00,&h00 '0108 YStep
[DU4] 392: $INLINE &h00,&h00 '010A Old_x
[LU4] 393: $INLINE &h00,&h00 '010C Old_y
[TB6] 394: $INLINE &h00 '010E Argument 5
(farbe%)
[7L4] 395: $INLINE &h53 '010F PUSH BX
[XH4] 396: $INLINE &h57 '0110 PUSH DI
[7M6] 397: $INLINE &h8A,&h3E,&h0E,&h01 '0111 MOV B
H,[010E]
[4B5] 398: $INLINE &h33,&hC0 '0115 XOR AX,AX
[H95] 399: $INLINE &h8A,&hC3 '0117 MOV AL,BL
[N24] 400: $INLINE &hD1,&hE0 '0119 SHL AX,1
[Y24] 401: $INLINE &hD1,&hE0 '011B SHL AX,1
[005] 402: $INLINE &hD1,&hE0 '011D SHL AX,1
[205] 403: $INLINE &hD1,&hE0 '011F SHL AX,1
[H24] 404: $INLINE &hD1,&hE0 '0121 SHL AX,1
[NB5] 405: $INLINE &h8B,&hF8 '0123 MOV DI,AX
[N24] 406: $INLINE &hD1,&hE0 '0125 SHL AX,1
[RZ4] 407: $INLINE &hD1,&hE0 '0127 SHL AX,1
[D75] 408: $INLINE &h03,&hF8 '0129 ADD DI,AX
[H85] 409: $INLINE &h03,&hFA '012B ADD DI,DX
[PH6] 410: $INLINE &hF7,&hC2,&h01,&h00 '012D TEST
DX,0001
[WU4] 411: $INLINE &h75,&h01 '0131 JNZ 0134
[CC4] 412: $INLINE &h47 '0133 INC DI
[9A6] 413: $INLINE &h26,&h8A,&h05 '0134 MOV AL,ES:
[DI]
[PG6] 414: $INLINE &hF7,&hC2,&h01,&h00 '0137 TEST
DX,0001
[1N4] 415: $INLINE &h74,&h04 '013B JZ 0141
[Z95] 416: $INLINE &h32,&hC7 '013D XOR AL,BH
[D24] 417: $INLINE &hEB,&h0A '013F JMP 014B
[U24] 418: $INLINE &hD0,&hE7 '0141 SHL BH,1
[W24] 419: $INLINE &hD0,&hE7 '0143 SHL BH,1
[Y24] 420: $INLINE &hD0,&hE7 '0145 SHL BH,1
[005] 421: $INLINE &hD0,&hE7 '0147 SHL BH,1
[S95] 422: $INLINE &h32,&hC7 '0149 XOR AL,BH
[W86] 423: $INLINE &h26,&h88,&h05 '014B MOV ES:[DI
],AL
[CF4] 424: $INLINE &h5F '014E POP DI
[GF4] 425: $INLINE &h5B '014F POP BX
[D24] 426: $INLINE &hC3 '0150 RET
[000] 427: '
[SK4] 428: $INLINE &h55 '0151 PUSH BP
[1L4] 429: $INLINE &h1E '0152 PUSH DS
[B05] 430: $INLINE &h8B,&hEC '0153 MOV BP,SP
[D66] 431: $INLINE &hC4,&h7E,&h08 '0155 LES DI,[BP
+08]
[8C6] 432: $INLINE &h26,&h8A,&h1D '0158 MOV BL,ES:
[DI]
[MC5] 433: $INLINE &h80,&hE3,&h0F '015B AND BL,0F
[SH6] 434: $INLINE &h88,&h1E,&h0E,&h01 '015E MOV [
010E],BL
[V86] 435: $INLINE &hC4,&h7E,&h0C '0162 LES DI,[BP
+0C]
[RC6] 436: $INLINE &h26,&h8B,&h1D '0165 MOV BX,ES:
[DI]
[8F6] 437: $INLINE &h89,&h1E,&h04,&h01 '0168 MOV [
0104],BX
[MD5] 438: $INLINE &h8B,&hCB '016C MOV CX,BX
[E56] 439: $INLINE &hC4,&h7E,&h10 '016E LES DI,[BP
+10]
[RC6] 440: $INLINE &h26,&h8B,&h1D '0171 MOV BX,ES:
[DI]
[TE6] 441: $INLINE &h89,&h1E,&h02,&h01 '0174 MOV [
0102],BX
[3E5] 442: $INLINE &h8B,&hF3 '0178 MOV SI,BX
[A66] 443: $INLINE &hC4,&h7E,&h14 '017A LES DI,[BP
+14]
[AD6] 444: $INLINE &h26,&h8B,&h1D '017D MOV BX,ES:
[DI]
[TD5] 445: $INLINE &h85,&hDB '0180 TEST BX,BX
[3W4] 446: $INLINE &h79,&h04 '0182 JNS 0188
[5N6] 447: $INLINE &h8B,&h1E,&h0C,&h01 '0184 MOV B
X,[010C]
[176] 448: $INLINE &hC4,&h7E,&h18 '0188 LES DI,[BP
+18]
[7C6] 449: $INLINE &h26,&h8B,&h15 '018B MOV DX,ES:
[DI]
[SD5] 450: $INLINE &h85,&hD2 '018E TEST DX,DX
[1W4] 451: $INLINE &h79,&h04 '0190 JNS 0196
```

```
[SL6] 452: $INLINE &h8B,&h16,&h0A,&h01 '0192 MOV D
X,[010A]
[846] 453: $INLINE &hA1,&h02,&h01 '0196 MOV AX,[01
02]
[N76] 454: $INLINE &hA3,&h0A,&h01 '0199 MOV [010A]
,AX
[Z46] 455: $INLINE &hA1,&h04,&h01 '019C MOV AX,[01
04]
[E86] 456: $INLINE &hA3,&h0C,&h01 '019F MOV [010C]
,AX
[EN5] 457: $INLINE &hB8,&h00,&hB8 '01A2 MOV AX,B80
0
[HD5] 458: $INLINE &h8E,&hC0 '01A5 MOV ES,AX
[6D5] 459: $INLINE &h33,&hC0 '01A7 XOR AX,AX
[EE5] 460: $INLINE &h2B,&hF2 '01A9 SUB SI,DX
[RE5] 461: $INLINE &h1D,&h00,&h00 '01AB SBB AX,000
0
[1B5] 462: $INLINE &h03,&hF0 '01AE ADD SI,AX
[Z55] 463: $INLINE &h33,&hF0 '01B0 XOR SI,AX
[1E5] 464: $INLINE &h0D,&h01,&h00 '01B2 OR AX,0001
[RD5] 465: $INLINE &h8B,&hF8 '01B5 MOV DI,AX
[LG6] 466: $INLINE &h89,&h36,&h06,&h01 '01B7 MOV [
0106],SI
[4G4] 467: $INLINE &h4E '01BB DEC SI
[4R4] 468: $INLINE &h78,&h02 '01BC JS 01C0
[UF5] 469: $INLINE &h33,&hF6 '01BE XOR SI,SI
[7D5] 470: $INLINE &h33,&hC0 '01C0 XOR AX,AX
[ND5] 471: $INLINE &h2B,&hCB '01C2 SUB CX,BX
[HE5] 472: $INLINE &h1D,&h00,&h00 '01C4 SBB AX,000
0
[R95] 473: $INLINE &h03,&hC8 '01C7 ADD CX,AX
[8E5] 474: $INLINE &h33,&hC8 '01C9 XOR CX,AX
[NE5] 475: $INLINE &h0D,&h01,&h00 '01CB OR AX,0001
[L76] 476: $INLINE &hA3,&h08,&h01 '01CE MOV [0108]
,AX
[PC6] 477: $INLINE &h39,&h16,&h02,&h01 '01D1 CMP [
0102],DX
[A05] 478: $INLINE &h75,&h08 '01D5 JNZ 01DF
[0E6] 479: $INLINE &h39,&h1E,&h04,&h01 '01D7 CMP [
0104],BX
[B05] 480: $INLINE &h75,&h02 '01DB JNZ 01DF
[115] 481: $INLINE &hEB,&h18 '01DD JMP 01F7
[0K5] 482: $INLINE &hE8,&h2D,&hFF '01DF CALL 010F
[LE5] 483: $INLINE &h83,&hFE,&h00 '01E2 CMP SI,+00
[3Y4] 484: $INLINE &h7D,&h0A '01E5 JGE 01F1
[4F6] 485: $INLINE &h03,&h1E,&h08,&h01 '01E7 ADD B
X,[0108]
[3E6] 486: $INLINE &h03,&h36,&h06,&h01 '01EB ADD S
I,[0106]
[N05] 487: $INLINE &hEB,&hE0 '01EF JMP 01D1
[D85] 488: $INLINE &h03,&hD7 '01F1 ADD DX,DI
[RE5] 489: $INLINE &h2B,&hF1 '01F3 SUB SI,CX
[915] 490: $INLINE &hEB,&hDA '01F5 JMP 01D1
[0H4] 491: $INLINE &h1F '01F7 POP DS
[4H4] 492: $INLINE &h5D '01F8 POP BP
[HA1] 493: END SUB
[MM0] 494: '
```

Listing 1. Die Include-Datei »cga.inc« enthält die Funktionen für den 16-Farben-Modus der CGA-Karte

```
[TB4] 1: 'Demo des CGA-Modus 160*100/16
[ST3] 2: '(C) 1989 Everts&Hagedorn
[W53] 3: 'DOS International
[PWE] 4: '
[000] 5: '
[DR2] 6: $include "cga.inc"
[FA3] 7: on error goto fehler
[CF4] 8: on timer(3) gosub ChangeRand
[AC1] 9: timer on
[TL1] 10: defint a-z
[US2] 11: randomize timer
[Z02] 12: call CgaInit
[NC2] 13: call CgaRand(1)
[3N2] 14: for Zeile=1 to 12
[817] 15: call CgaPrint("Dies ist ein Demo",2,Zeile
,1)
[HT1] 16: next Zeile
[Y41] 17: delay 1
[472] 18: call CgaCls(0)
[K83] 19: for y2=0 to 99 step 3
[0E1] 20: f=rnd*14+1
```



```

[HT3] 21: call CgaLine(0,0,159,y2,f)
[D14] 22: call CgaLine(0,99,159,y2,f)
[ZB4] 23: call CgaLine(159,99,0,99-y2,f)
[R34] 24: call CgaLine(159,0,0,99-y2,f)
[K91] 25: next y2
[Y41] 26: delay 1
[472] 27: call CgaCls(0)
[A22] 28: for n=0 to 100
[TN1] 29: x1=rnd*158+1
[WN1] 30: x2=rnd*158+1
[9M1] 31: y1=rnd*98+1
[CM1] 32: y2=rnd*98+1
[OE1] 33: f=rnd*14+1
[PT4] 34: call CgaXorLine(x1,y1,x2,y2,f)
[C91] 35: delay .05
[PT4] 36: call CgaXorLine(x1,y1,x2,y2,f)
[921] 37: next n
[Y41] 38: delay 1
[472] 39: call CgaCls(0)
[E22] 40: for n=0 to 500
[TN1] 41: x1=rnd*158+1
[WN1] 42: x2=rnd*158+1
[9M1] 43: y1=rnd*98+1
[CM1] 44: y2=rnd*98+1
[OE1] 45: f=rnd*14+1
[W74] 46: call CgaLine(x1,y1,x2,y2,f)
[921] 47: next n
[Y41] 48: delay 1
[472] 49: call CgaCls(0)
[S66] 50: call CgaPrint("Weiter mit Taste",14,1,1)
[NB0] 51: do
[FB3] 52: zaehler=ceil(rnd*100)
[RB4] 53: x=int(rnd*160):y=int(rnd*100)
[4T2] 54: f=(zaehler mod 15)+1
[9W3] 55: call CgaXorPixel(x,y,f)
[NB0] 56: do
[B22] 57: incr zaehler
[434] 58: zaehler=zaehler mod 32000
[MF3] 59: dirx=sgn(int(rnd*3)-1)
[NF3] 60: diry=sgn(int(rnd*3)-1)
[SE3] 61: nx=x+dirx:ny=y+diry
[C24] 62: if nx<1 or nx>158 then exit loop
[HW4] 63: if ny<0 or ny>99 then exit loop
[AC4] 64: call CgaPoint(nx,ny,farbe)
[1U2] 65: select case farbe
[MS1] 66: case >8
[622] 67: exit select
[AX0] 68: case 0
[8D2] 69: diry=sin(diry)
[DX0] 70: case 1
[8X2] 71: dirx=cos(zaehler)
[GX0] 72: case 2
[9X2] 73: diry=cos(zaehler)
[LX0] 74: case 3
[8X2] 75: dirx=cos(zaehler)
[WX2] 76: diry=sin(zaehler)
[PX0] 77: case 4
[VX2] 78: dirx=sin(zaehler)
[TX0] 79: case 5
[8X2] 80: dirx=cos(zaehler)
[WX0] 81: case 6
[WX2] 82: diry=sin(zaehler)
[ZX0] 83: case 7
[9X2] 84: diry=cos(zaehler)
[2Y0] 85: case 8
[VX2] 86: dirx=sin(zaehler)
[9X2] 87: diry=cos(zaehler)
[GS1] 88: end select
[SH1] 89: nx=x+dirx
[YH1] 90: ny=y+diry
[C24] 91: if nx<1 or nx>158 then exit loop
[HW4] 92: if ny<0 or ny>99 then exit loop
[RR3] 93: f=((zaehler/100) mod 15)+1
[AC4] 94: call CgaPoint(nx,ny,farbe)
[LM6] 95: if farbe=0 then call CgaXorPixel(nx,ny,f)
[WS1] 96: nx=nx+dirx
[AC4] 97: call CgaPoint(nx,ny,farbe)
[LM6] 98: if farbe=0 then call CgaXorPixel(nx,ny,f)
[AL1] 99: x=nx:y=ny
[922] 100: loop until instat
[922] 101: loop until instat
[Y41] 102: delay 1
[N42] 103: dummy$=inkey$
[472] 104: call CgaCls(0)
[3K0] 105: fl=0
[NUS] 106: call CgaPrint("Abbruch = Taste",14,6,3)
[NB0] 107: do
[7L0] 108: x1=0
[CM0] 109: f=f1
[1U1] 110: add=rnd*15+1
[NB0] 111: do
[SR0] 112: x=x1
[AZ1] 113: for y=0 to 99
[9W3] 114: call CgaXorPixel(x,y,f)
[B94] 115: call CgaXorPixel(x,99-y,f)

```

```

[211] 116: incr x
[ZS0] 117: next
[CZ0] 118: incr f
[PH1] 119: f=f mod 16
[671] 120: incr x1
[623] 121: loop until x1+99>159
[UT1] 122: incr fl:add
[1N2] 123: fl=f1 mod 16-(f1=14)
[4L2] 124: if rnd*100>90 then
[CE2] 125: call CgaCls(f1)
[NUS] 126: call CgaPrint("Abbruch = Taste",14,6,3)
)
[801] 127: end if
[922] 128: loop until instat
[000] 129:
[F91] 130: fehler:
[8R1] 131: screen 0,0,0
[SE2] 132: on error goto 0
[4M0] 133: end
[000] 134:
[621] 135: ChangeRand:
[5M1] 136: fa=rnd*15+1
[XK2] 137: call CgaRand(fa)
[651] 138: return

```

Listing 2. Das Programm »demo.bas« führt die einzelnen Funktionen der Turbo-Basic-Bibliothek vor

```

[953] 1: rem CgaFraktal für
[CP2] 2: rem Turbo Basic
[A95] 3: rem im CGA-Spezialmodus 160*100/16
[YE7] 4: rem (C) 1989 Redaktionsbüro Everts&Hagedorn
[FT3] 5: rem DOS International
[000] 6:
[BT4] 7: 'Fraktal im CGA-Modus 160*100/16
[4G3] 8: '-----
[000] 9:
[DR2] 10: $include "cga.inc"
[FA3] 11: on error goto fehler
[Z02] 12: call CgaInit
[1D2] 13: call CgaRand(5)
[H56] 14: defdbl a-z 'kann ohne 8087 entfallen !
[2Y0] 15: %kx=50
[9P0] 16: %s=20
[2Y1] 17: xu=-.7:xo=2.3
[WC2] 18: yu=-1.25:yo=1.25
[332] 19: dx=(xo-xu)/160
[X22] 20: dy=(yo-yu)/100
[KV0] 21: yc=yu
[362] 22: for m%=0 to 99
[FV0] 23: xc=xu
[Z92] 24: for n%=0 to 159
[TR0] 25: x=-xc
[WR0] 26: y=-yc
[EH2] 27: for k%=1 to %kx
[XB1] 28: xalt=x+x
[FB2] 29: x=(x+y)*(x-y)-xc
[G02] 30: y=xalt*y-yc
[CD4] 31: if x*x+y*y>%s then exit for
[M61] 32: next k%
[CL2] 33: k%=k% and (k%<=%kx)
[VBA] 34: call CgaPixel(n%,m%,k% and 15)
[AR1] 35: incr xc,dx
[X61] 36: next n%
[8Z3] 37: if instat then exit for
[GR1] 38: incr yc,dy
[U61] 39: next m%
[NB0] 40: do
[922] 41: loop until instat
[000] 42:
[F91] 43: fehler:
[8R1] 44: screen 0,0,0
[SE2] 45: on error goto 0
[4M0] 46: end

```

Listing 3. »cgafrakt.bas« zeichnet das Apfelmännchen-Fraktal in 16 Farben

Fensterorientierte Funktionen selbst programmiert

Bislang kostete es viel Zeit und Mühe, in GW-Basic oder SBasic fensterorientierte Programme zu schreiben. Ab sofort nehmen Ihnen die vier Funktionen des Listings »screen.bas«, viel Arbeit ab.

Fensterorientierte Applikationen liegen im Trend, weil Sie benutzerfreundlich sind. Deshalb sollten auch Sie sich dieser Entwicklung nicht verschließen. Doch das Programmieren von Bildschirmfenstern ist aufwendig und nicht jedermanns Sache. Abhilfe schafft die Funktionssammlung von »screen.bas« (Listing), die Ihnen viel Arbeit erspart. Alle Funktionen sind als schnelle Maschinencode-Unterprogramme ausgeführt und werden über den Basic-Befehl »call« aufgerufen.

Das Programm »screen.bas« enthält neben den Daten der Maschinencode-Routinen und dem Initialisierungsteil auch eine kleine Demonstration der einzelnen Funktionen (Programmzeile 200 bis 970), die Sie selbstverständlich entfernen können, wenn Sie sie nicht benötigen.

Die Funktion »fill« füllt ein gewünschtes Bildschirmfenster mit einem Zeichen oder Attribut Ihrer Wahl. Die Syntax der Fill-Funktion lautet:

```
call fill (x%, y%, b%, h%, z%, a%)
```

Die Integer-Variable »x%« gibt die erste Textspalte und »y%« die erste Textzeile des zu füllenden Bildschirmfensters (jeweils ab 0) an. Über die beiden Variablen »b%« und »h%« bestimmen Sie die Breite und Höhe des zu füllenden Fensters. Die Variable »z%« enthält das Zeichen, mit dem dieser Bereich gefüllt werden soll, und »a%« das Attribut. Wollen Sie kein Zeichen oder Attribut angeben, so weisen Sie der entsprechenden Variablen den Wert -1 zu. Die Funktion »invert« invertiert ein gewünschtes Bildschirmfenster. Die

Syntax dieser Funktion lautet

```
call invert (x%, y%, b%, h%)
```

Die Bedeutung der Parameter ist die gleiche, wie die der gleichnamigen Parameter bei der Fill-Funktion. Die Attributänderung erfolgt bei der Invert-Funktion nach folgendem Muster:

Bit 3 (Leuchtend) und Bit 7 (Blinkend) bleiben unverändert, und die Bit 0 bis 2 (Vordergrund) sowie die Bit 4 bis 6 (Hintergrund) werden miteinander vertauscht. Folgende Kombinationen sind für diese beiden Bitbereiche relevant (im Format »Vordergrund/Hintergrund«):

000/000: Zeichen unsichtbar
001/000: unterstrichene Darstellung
111/000: normale Darstellung
000/111: inverse Darstellung

»screen.bas« liefert vier leistungsfähige Funktionen zur Fenster-Verwaltung in GW-Basic und SBasic

Die Funktion »move« sichert und restauriert einen definierten Bildschirmbereich. Dies ist besonders für den Ausschnitt hinter sich öffnenden und schließenden Fenstern interessant.

Die Syntax der Move-Funktion lautet:

```
call move (x%, y%, b%, h%, p%(0), m%)
```

Zuvor muß allerdings mit

```
dim p% (b% * h%)
```

ein Array »p%()« mit Breite-mal-Höhe-Feldern als Puffer für das Bildschirm-

fenster definiert (jedes Element in dem Array »p%« besteht aus einem Zeichen und dem zugehörigem Attribut) und mit

m% = Modus

der Betriebsmodus der Move-Funktion festgelegt werden.

In Abhängigkeit von dem Wert der Variablen »m%« führt die Move-Funktion folgende Aktion durch:

- m% = 0: Bildschirmfenster in den Puffer »p%()« kopieren
- m% = 1: Pufferinhalt von »p%()« in das Bildschirmfenster kopieren
- m% = 2: Inhalt des Bildschirmfensters und des Puffers miteinander vertauschen

Die Funktion »frame« zeichnet um ein definiertes Bildschirmfenster einen einfachen oder doppelten Rahmen. Die Syntax dieser Funktion lautet:

```
call frame (x%, y%, b%, h%, m%, a%)
```

Die Integer-Variable »m%« gibt durch ihren Wert an, ob der Rahmen einfach (m% = 0) oder doppelt gezogen werden soll (m% = 1). Die Breite und die Höhe des Rahmens muß mindestens zwei Zeichenpositionen betragen. Beim Attribut ist die Angabe von -1 nicht zulässig. Die in Frage kommenden Attribute für monochrome Bildschirmadapter und Farbgrafikkarten können Sie jedoch den Tabellen 1 und 2 entnehmen.

Die vier Funktionen von »screen.bas« funktionieren übrigens nur im 80-Zeichen-Modus. Es werden allerdings mehrere Bildschirmseiten unterstützt. Sind die Parameter bei einer Funktion so gewählt, daß das Fenster ganz oder teilweise außerhalb des Bildschirms liegen würde, so wird überhaupt keine Aktion durchgeführt.

(Wolfgang Lorenz/ma)

Wert	Funktion
0	unsichtbar
1	unterstrichen
7	normal
9	leuchtend und unterstrichen
15	leuchtend
112	invers
129	blinkend und unterstrichen
135	blinkend
137	blinkend, leuchtend und unterstrichen
143	blinkend und leuchtend
240	blinkend und invers

Tabelle 1. Die verschiedenen Attribute für monochrome Bildschirmadapter

Name:	screen.bas
Sprache	GW-Basic, SBasic
Grafikkarte	jede
Besonderheiten	nur Textmodus

Bit	Bedeutung
0	Blau-Anteil für die Vordergrundfarbe
1	Grün-Anteil für die Vordergrundfarbe
2	Rot-Anteil für die Vordergrundfarbe
3	Intensitäts-Anteil für die Vordergrundfarbe
4	Blau-Anteil für die Hintergrundfarbe
5	Grün-Anteil für die Hintergrundfarbe
6	Rot-Anteil für die Hintergrundfarbe
7	Intensitäts-Anteil für die Hintergrundfarbe oder blinkende Darstellung (Steuerung über Bit 5 des Modus-Auswahlregisters unter der Portadresse 3D8hex)

Tabelle 2. Der Aufbau des Attribut-Byte für Farbgrafikkarten


```
[C5B] 100 MC0$=SPACE$(212):FOR I%=1 TO 105:READ X$:MID$(
(MC0$,I%*2+1,2)=MKI$(X%):NEXT
[Z1B] 101 MC1$=SPACE$(198):FOR I%=0 TO 98:READ X$:MID$(
MC1$,I%*2+1,2)=MKI$(X%):NEXT
[AS8] 102 INVERT=PEEK(VARPTR(MC0$)+1)+PEEK(VARPTR(MC0$)
+2)*256+2
[FX2] 103 FILL=INVERT+48
[X53] 104 MOVE=INVERT+119
[V48] 105 FRAME=PEEK(VARPTR(MC1$)+1)+PEEK(VARPTR(MC1$)+
2)*256
[W15] 106 MID$(MC0$,1,2)=MKI$(FRAME-65433!)
[5FA] 107 DATA &HEC8B,&HE806,&H0000,&HFF5B,&HF857,&H207
2,&H00B5,&HC8BA,&H2647,&H058A
[PPA] 108 DATA &HE08A,&H7725,&HD088,&HD0C0,&HD0C0,&HD0C
0,&H0AC0,&HAAC4,&HEAE2,&HFA03
[APA] 109 DATA &HCFFE,&HE275,&HCA07,&H0008,&HEC8B,&HC58
3,&H0604,&H00E8,&H5B00,&H57FF
[5BA] 110 DATA &H72C5,&H8B34,&H0276,&H8BAD,&H0076,&H0C8
B,&HE422,&H1275,&HED22,&H1575
[SVA] 111 DATA &HE18A,&HC8BA,&HABF3,&HFA03,&HCFFE,&HF67
5,&H15EB,&HED22,&H1175,&HC18A
[VSA] 112 DATA &HB547,&H8A00,&HAACB,&HE247,&H03FC,&HFEF
A,&H75CF,&H07F4,&H0CCA,&H8B00
[EKA] 113 DATA &H83EC,&H04C5,&HE806,&H0000,&HFF5B,&H7E9
7,&H72FF,&H8B47,&H0076,&H8BAD
[V9A] 114 DATA &H0276,&H348B,&H00B5,&H013C,&H1774,&H217
7,&HCB8A,&H8B26,&H4705,&H8947
[TXA] 115 DATA &H4604,&HE246,&H03F5,&HFEFA,&H75CF,&HEBE
D,&H8A21,&HFF3CB,&H03A5,&HFEFA
[KNA] 116 DATA &H75CF,&HEBF6,&H3C15,&H7502,&H8A11,&H26C
B,&H058B,&H89A5,&HFE44,&HF7E2
[6RA] 117 DATA &HFA03,&HCFFE,&HEF75,&HCA07,&H000C,&HEC8
B,&HC583,&H0604,&H5EE8,&H7200
[TEA] 118 DATA &H8158,&H02EB,&HF702,&H80C3,&H7580,&H8B4
E,&H0076,&H22AD,&H75E4,&H8A46
[NRA] 119 DATA &H8BE0,&H0276,&H3CAC,&H7701,&HAC3C,&H0EE
8,&HDA00,&HBFCA,&H0CB3,&HD9C4
[6TA] 120 DATA &HCDC9,&HBAB,&HCDC8,&H5EBC,&H0375,&HC68
3,&H2207,&H75C0,&HFE20,&H5C7
[22B] 121 DATA &H8A00,&HACCB,&HACAB,&HABF3,&HABAC,&HCB8
A,&H03AC,&HFEFA,&H74CF,&H78F0
[WHA] 122 DATA &HAB08,&HF903,&HF903,&HEBAB,&H07F0,&H0CC
A,&HB400,&HCD0F,&HB310,&H8000
[CKA] 123 DATA &HB8C7,&H073C,&H0275,&HB0B7,&HC38E,&H8BF
C,&H0A7E,&H158B,&HFA83,&H7350
[4FA] 124 DATA &H8B41,&H067E,&H1D8B,&HDB23,&H3874,&HC38
B,&HC203,&H3272,&H503D,&H7700
[RGa] 125 DATA &H8B2D,&H087E,&H058B,&H193D,&H7300,&H8B2
3,&H047E,&H0DB8,&H1CE3,&HF98A
[BEA] 126 DATA &HC803,&H1672,&HF983,&H7719,&HB411,&HF65
0,&H03E4,&HD1C2,&H8BE0,&HB2F8
[YY4] 127 DATA &H2A50,&HD0D3,&HC3E2,&HC3F9
[BS3] 200 '*** Demonstration ***
[G22] 210 DEFINT A-Z
[G41] 220 CLS
[FC2] 230 FOR I=0 TO 11
[V71] 240 X=I*3
[X71] 250 Y=I+1
[DL1] 260 B=80-X*2
[CL1] 270 H=25-Y*2
[W11] 280 Z=-1
[6N2] 290 A=7+105*(Y AND 1)
[YU3] 300 CALL FILL!(X,Y,B,H,Z,A)
[S71] 310 NEXT
[P73] 320 RANDOMIZE TIMER
[M58] 330 PRINT "Bildschirmbereiche füllen (nur Zeichen
)"
```

```
[9W2] 340 WHILE INKEY$=""
[VU1] 350 GOSUB 920
[EX2] 360 Z=INT(RND(1)*256)
[901] 370 A=-1
[PV3] 380 CALL FILL!(X,Y,B,H,Z,A)
[A71] 390 WEND
[W42] 400 LOCATE 1,27
[BA4] 410 PRINT "(nur Attribute)"
[5W2] 420 WHILE INKEY$=""
[RU1] 430 GOSUB 920
[M11] 440 Z=-1
[EV2] 450 A=INT(RND(1)*128)
[KV3] 460 CALL FILL!(X,Y,B,H,Z,A)
[671] 470 WEND
[M52] 480 LOCATE 1,27
[5P5] 490 PRINT "(Zeichen und Attribute)"
[1W2] 500 WHILE INKEY$=""
[LU1] 510 GOSUB 920
[6X2] 520 Z=INT(RND(1)*256)
[AV2] 530 A=INT(RND(1)*128)
[EV3] 540 CALL FILL!(X,Y,B,H,Z,A)
[271] 550 WEND
[314] 560 PRINT CHR$(11);SPACE$(80)
[MY1] 570 LOCATE 1,1
[VU6] 580 PRINT "Bildschirmbereiche invertieren"
[W01] 590 X=0
[901] 600 Y=1
[U01] 610 B=40
[411] 620 H=23
[N11] 630 Z=-1
[Z71] 640 A=H70
[LV3] 650 CALL FILL!(X,Y,B,H,Z,A)
[N21] 660 X=40
[601] 670 A=7
[WV3] 680 CALL FILL!(X,Y,B,H,Z,A)
[001] 690 I=0
[5W2] 700 WHILE INKEY$=""
[C31] 710 X=I
[H01] 720 Y=1
[6Z0] 730 B=1
[C11] 740 H=23
[NS3] 750 CALL INVERT!(X,Y,B,H)
[YC1] 760 X=I+40
[VS3] 770 CALL INVERT!(X,Y,B,H)
[G71] 780 I=I+1
[XT2] 790 IF I=40 THEN I=0
[T61] 800 WEND
[U04] 810 PRINT CHR$(11);SPACE$(80)
[BY1] 820 LOCATE 1,1
[M84] 830 PRINT "Rahmen zeichnen"
[LW2] 840 WHILE INKEY$=""
[V51] 850 GOSUB 920
[6R2] 860 M=INT(RND(1)*2)
[WV2] 870 A=INT(RND(1)*128)
[G64] 880 CALL FRAME!(X,Y,B,H,M,A)
[M71] 890 WEND
[S41] 900 CLS
[931] 910 END
[986] 920 '*** Zufallsausschnitt erzeugen ***
[AU2] 930 X=INT(RND(1)*61)
[T13] 940 Y=INT(RND(1)*15)+1
[I13] 950 B=INT(RND(1)*19)+2
[ZV2] 960 H=INT(RND(1)*8)+2
[XM1] 970 RETURN
```

Listing. Mit »screen.bas« lassen sich fensterorientierte Anwendungen wesentlich leichter programmieren

Wer — wo — wann?

Bei Zuschriften bitten wir, folgendes zu beachten: Haben Sie Fragen, Anregungen oder Programme, die Sie der DOS-Redaktion zukommen lassen möchten, adressieren Sie wie folgt:

Redaktionsbüro Everts & Hagedorn · Redaktion DOS International
Hans-Stießberger-Str. 3 · 8013 Haar/München · Tel. (089) 46 10 01-0

Haben Sie Fragen, die zum Beispiel Abonnements, Kleinanzeigen oder Databox-Bestellungen betreffen, wenden Sie sich bitte an den Verlag:

DMV-Verlag · Postfach 250 · 3440 Eschwege

Dauerhafte Funktionstastenbelegung in GW-Basic

Der Basic-Interpreter unterstützt den Programmierer nach dem Start mit einer vordefinierten Funktionstastenbelegung. Leider entspricht diese Belegung nicht immer den eigenen Wünschen. Dabei ist es gar nicht so schwer, sich seine eigene Standardbelegung zu definieren, die sofort nach dem Laden von GW-Basic aktiv ist.

Mit der vorgegebenen Funktionstastenbelegung, die die Entwickler von GW-Basic ihrem Produkt und damit allen Anwendern dieser Sprache mit auf den Weg gaben, ist nicht jeder Programmierer zufrieden. Somit wird man immer wieder gezwungen, in jedem neu geschriebenen Basic-Programm zuerst einmal die Belegung der Funktionstasten den persönlichen Erfordernissen anzupassen. Die Mühe der ständigen Neuebelegung von Funktionstasten ersparen Sie sich, wenn Sie Ihre eigene Tastendefinition direkt im Programmcode des GW-Basic-Interpreters festhalten. Fertigen Sie dazu zuerst einmal eine Kopie des GW-Basic-Interpreters an, der Sie den Namen »basic1.exe« zuweisen:

```
copy gwbasic.exe basic1.exe
```

Starten Sie als nächstes das Programm »Basic1«, und notieren Sie sich die aktuelle Funktionstastenbelegung. Legen Sie anschließend eine eigene Tastenbelegung fest. Für Programmentwickler dürfte eine Definition der Tasten, wie sie die Tabelle zeigt, vorteilhaft sein.

Mit [F5] sichern Sie nun zwischen durch – durch Drücken nur einer Taste – Ihr Programm auf Diskette oder Festplatte, ohne sich um den Programmnamen kümmern zu müssen;

die Umbenennung der Datei »temp.bas« erledigen Sie am Schluß Ihrer Programmierarbeit mit dem Befehl »rename«. Der Doppelpunkt am Ende der Zuweisung für [F5] ist keinesfalls ein Fehler, sondern wurde dort absichtlich platziert. Sie dürfen somit am Anfang jeder Bildschirmzeile diese Funktionstaste betätigen; selbst wenn dort Programmtext steht, erfolgt kein »Syntax Error«, weil der Doppelpunkt Basic-Befehle trennt.

Mit Ihrer persönlichen Funktionstastenbelegung machen Sie den Basic-Interpreter anwendungsfreundlicher

Mit der Funktionstaste [F7] initialisieren Sie nach dem erstmaligen Aufruf des Interpreters den Bildschirm. Die angedeuteten Punkte sollen deutlich machen, daß hier die 16 Byte, die für eine Funktionstastenbelegung reserviert sind, mit Leerzeichen aufgefüllt werden müssen.

Normalerweise lassen sich pro Funktionstaste nur 15 Zeichen zuordnen. Das 16te Byte ist 00hex und dient als Trennzeichen zur nächsten Tastenbelegung. In diesem Fall führt jedoch das vollständige Auffüllen mit Leerzeichen dazu, daß gleich anschließend der Inhalt der Taste [F8] ausgeführt wird.

Leider reicht der Platz zur Belegung dieser Funktionstaste nicht aus, um noch einen Doppelpunkt hinter dem Befehl »cls« aufzunehmen. [F7] und [F8] dürfen Sie daher nur dann betätigen, wenn der Cursor in leeren Bildschirmzeilen steht.

Für die Definition einer Funktionstastenbelegung stehen jeweils 16 Byte zur Verfügung

Mit Hilfe der Funktionstaste [F9] realisieren Sie nun einen temporären Aufruf von MS-DOS, das heißt, Sie befinden sich anschließend auf der Betriebssystemebene und dürfen alle Systemkommandos ausführen. Um jedoch zurück in den Interpreter zu gelangen, müssen Sie diesen nicht erneut aufrufen, sondern nur das

Kommando »exit« ausführen – das geht wesentlich schneller. Mit [F10] verlassen Sie GW-Basic endgültig und gelangen in das Betriebssystem MS-DOS zurück.

Damit die Eingabe fehlersicher wird, ist nachstehend die Funktionstastenbelegung in hexadezimaler Schreibweise für alle 16 Byte des jeweiligen Speicherbereichs angegeben:

```
F5 - 53 41 56 45 22 54 45 4D 50 22 2C 41 0D
00 00 00
F7 - 4B 45 59 20 4F 46 46 3A 4E 45 57 3A 0D
20 20 20
F8 - 43 4F 4C 4F 52 20 31 34 2C 31 3A 43 4C
53 0D 00
F9 - 53 48 45 4C 4C 3A 0D 00 00 00 00 00 00
00 00 00
F10 - 43 4C 53 3A 53 59 53 54 45 4D 3A 0D 00
00 00 00
```

Die Belegung der Taste [F7] mit »key off new:[Enter]« eignet sich nicht für »zerstreute« Programmierer, da bei einem versehentlichen Tastendruck während des Programmierens das gesamte Programm im Hauptspeicher verlorengeht. Alle, die sich einer solchen Gefahr nicht aussetzen und lieber ihre Nerven schonen wollen, finden daher nachfolgend eine Fassung, die auf eine Bestätigung durch die Enter-Taste wartet:

```
F7 - 4B 45 59 20 4F 46 46 3A 4E 45 57 3A 00
00 00 00
```

Zu beachten ist, daß die Abstände von 16 Byte strikt eingehalten werden müssen, da der Interpreter den Inhalt der Funktionstastenbelegung genau ab dieser Grenze liest und auf den Bildschirm schreibt.

Um die Änderungen an der Kopie Ihres Basic-Interpreters durchzuführen und die geänderte Version anschließend zu speichern, ist der MS-DOS-Debugger »debug« leider nicht geeignet, weil er nur Dateien mit weniger als 64 KByte Speicherbelegung verkraftet. Der Interpreter »gwbasic.exe« ist im Normalfall jedoch größer als 64 KByte. Sie müssen daher auf einen Byte-orientierten Diskettenmonitor zurückgreifen, wie ihn beispielsweise die »Norton Utilities« oder »PC-Tools« liefern. Mit der Funktion »search« beziehungsweise »find« suchen Sie damit die Texte der alten Funktionstastenbelegung und überschreiben diese anschließend mit dem neuen Inhalt.

Die angegebene Methodik der dauerhaften Neuebelegung von Funktionstasten funktioniert mit allen Basic-Interpreter-Versionen von »basic.com« und »gwbasic.com«.

(Peter Graetsch/zi)

Funktions-taste	Belegung
F1	list (unverändert)
F2	run [Enter] (unverändert)
F3	load" (unverändert)
F4	save" (unverändert)
F5	save"temp",a: [Enter]
F6	,"lpt1:" (unverändert)
F7	key off:new: [Enter] ...
F8	color 14,1:cls [Enter]
F9	shell: [Enter]
F10	cls:system: [Enter]

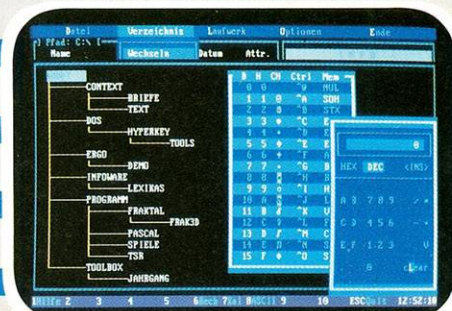
Tabelle. So kann die neue Belegung der Funktionstasten Ihres Basic-Interpreters aussehen

Einer für alles!

ERGO



Die Benutzeroberfläche, die auch "Profis" alles bietet und mit 12 kByte so gut wie keinen Speicherplatz beansprucht.



Technische Details:

- **Pulldownmenüs**
- **Dateiverwaltung**
kopieren · löschen · wiederherstellen (undelete) · suchen · umbenennen · verstecken · schützen · verschieben · Attribute setzen · formatieren · drucken
- **Verzeichnisverwaltung**
kopieren · löschen · verschieben (umhängen) · umbenennen · verstecken · schützen · Attribute setzen · Baumstruktur
- **Arbeitshilfen**
Kalender · Taschenrechner hex und dezimal · ASCII-Tabelle · Uhr · Notizblock
- **Informationssystem**
kontextbezogene Hilfeaufrufe · Hardwareinformationen · Massenspeicherbelegung (Mapping)
- **Menügenerator**
um eigene Menüpunkte erweiterbar · komfortable Pulldown-technik · ohne Programmierkenntnisse zu verwirklichen
- **Editor**
Fullscreen-Text (Wordstar-kompatibel) · Hex-Editor mit Suchfunktionen · Sektor-Editor für Massenspeicher
- **Paßwortvergabe mit Zugriffshierarchie**

Notwendige Konfiguration:

PC XT/AT mit 256 kByte oder mehr, 1 Diskettenlaufwerk bzw. Festplatte, MS-DOS ab Version 3.0



... und alles auch mit der Maus zu bedienen!

149,- DM (unverbindliche Preisempfehlung)

Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		Ausland:	
Einzelpreis	149,- DM	Einzelpreis	149,- DM
zzgl. Versandkosten	3,- DM	zzgl. Versandkosten	5,- DM
Endpreis	152,- DM	Endpreis	154,- DM

Best.-Nr.: 252 5 1/4"-Disk. Best.-Nr.: 253 3 1/2"-Disk.

Bitte benutzen Sie die Bestellkarte

DMV-Verlag · Postfach 250 · 3440 Eschwege

Noch immer nicht überzeugt? Dann sollten Sie unsere ERGO-Demo-Diskette für nur 5,- DM anfordern (Vorkasse)

Basic-Funktionen zur String-Verwaltung

Die Verwaltung von Zeichenketten in Basic ist ein nicht ganz einfaches, aber dafür umso interessanteres Thema. Die folgende Basic-Erweiterung »find.bas« stellt Ihnen unter GW-Basic und SBasic drei leistungsfähige String-Funktionen zur Verfügung.

Für textorientierte Anwendungen allgemein und speziell für die Übergabe von Strings an Maschinencode-Unterprogramme in SBasic ist die Funktionssammlung der Basic-Erweiterung »find.bas« (Listing) gedacht. Sie stellt für diesen Zweck drei neue Funktionen bereit. Alle Funktionen sind als schnelle Maschinencode-Unterprogramme ausgeführt und werden über den Basic-Befehl »call« aufgerufen.

Neben den Daten der Maschinensprache-Routinen und dem Initialisierungsteil enthält »find.bas« eine kurze Demonstration der speziellen Funktion »find« (Programmzeile 200 bis 430).

Die Funktion »convert« wandelt Zeichenketten komplett in Groß- oder Kleinschreibung um. Die Syntax für diese Funktion lautet:

```
call convert(s$,m%)
```

Die String-Variable »s\$« bezeichnet die umzuwandelnde Zeichenkette und die Integer-Variable »m%« gibt mit ihrem Wert an, ob Kleinschreibung (m% = 0) oder Großschreibung (m% = 1) für »s\$« erwünscht ist. Auch die deutschen Umlaute werden von der Convert-Funktion richtig umgewandelt, nur das Zeichen »ß« bleibt unverändert.

Beachten Sie bei einer Zuweisung in der Art

```
s$ = "text"
```

und der darauffolgenden Konvertierung von »s\$«, daß die Umwandlung direkt im Programm vorgenommen wird, das heißt, die Funktion »convert« modifiziert die String-Variable direkt im Quelltext, wie Sie beim Listen des Programms erkennen können.

Ist diese Form der Umwandlung unerwünscht, dann erzwingen Sie durch die Zuweisung

```
s$ = "text" + ""
```

daß die Zeichenkette im Basic-Datenbereich angelegt wird.

Die Funktion »borders« ermittelt die Grenzen einer Zeichenkette innerhalb einer mit Leerzeichen aufgefüllten String-Variablen. Dazu sucht die Borders-Funktion ab einer zu definierenden Anfangsposition über eine gewünschte Länge nach dem ersten

und letzten Zeichen, das kein Leerzeichen ist.

Die Syntax für die Borders-Funktion lautet:

```
call borders(s$,a%,l%)
```

Die String-Variable »a\$« bezeichnet wieder die zu bearbeitende Zeichenkette, »a%« gibt die Anfangsposition in der Zeichenkette an, ab der gesucht werden soll und »l%« enthält die Länge, über die die Zeichenkette untersucht werden soll. Wenn Sie

```
a%=1
```

```
und
```

```
l%=255
```

definieren, wird grundsätzlich die gesamte Zeichenkette nach den Grenzen durchsucht.

Drei neue Funktionen zur String-Verwaltung unter GW-Basic und SBasic liefert »find.bas«

Nach dem Aufruf der Borders-Funktion enthalten die Variablen »a%« und »l%« die vordere und hintere Grenze der gewünschten Teilzeichenkette ohne Leerzeichen. Die Zeichenkette läßt sich nun mit

```
x$ = mid$(s$, a%, l%)
```

weiterverarbeiten. Dieses Verfahren bietet sich beispielsweise für die Übergabe von mehreren String-Parametern an Maschinencode-Unterprogramme in SBasic an.

Die Funktion »find« sucht in einer String-Variablen, in einem String-Array oder in einem bestimmten Ausschnitt eines String-Array nach der gewünschten Zeichenkette. Dabei wird die Groß- und Kleinschreibung wahlweise berücksichtigt oder ignoriert.

Die Syntax für die Find-Funktion lautet:

```
call find(s$, f$, n%, m%, e%)
```

```
oder
```

```
call find(s$, f$(i%), n%, m%, e%)
```

Die String-Variable »s\$« enthält die zu suchende Zeichenkette, »f\$« gibt die zu durchsuchende String-Variable und »f\$(i%)« das zu durchsuchende String-Array an, wobei »i%« das erste relevante Array-Element und »n%« die Gesamtzahl der zu berücksichtigenden Elemente bezeichnet.

Über die Integer-Variable »m%« können Sie festlegen, ob bei der Suche die Groß- und Kleinschreibung berücksichtigt (m% = 0) oder ignoriert wird (m% = 1).

Nach dem Aufruf der Find-Funktion enthält die Variable »e%« den Offset des Array-Elements bezüglich »i%«, in dem der Inhalt der Variablen »s\$« gefunden wurde. Der Vergleich war in diesem Fall also beim Array-Element »f\$(i% + e%)« erfolgreich.

Durchsucht die Find-Funktion kein Array, sondern nur eine einzelne String-Variable, so muß beim Aufruf »n%« auf 1 gesetzt sein und die Variable »e%« enthält bei erfolgreicher Suche den Wert 0, bei Mißerfolg den Wert -1.

Die Find-Funktion arbeitet auch mit den MS-DOS-Jokern »?« und »*«

In der zu suchenden Zeichenkette »s\$« können Sie zwei verschiedene Joker einsetzen, die Ihnen bereits von MS-DOS her bekannt sein dürften und folgende Bedeutung haben:

»?«: steht für ein beliebiges einzelnes Zeichen.

»*«: steht für eine beliebige Anzahl (auch 0) beliebiger Zeichen.

Dazu ein Beispiel: Wenn Sie als zu suchende Zeichenketten beispielsweise »ABC«, »A?C«, »???«, »*«, »A*«, »*C«, »A*C«, »*B*«, »*ABC*« oder »*A***?C*« angeben, so wird von diesen Bezeichnungen die Kombination »ABC« gefunden, »CBA«, »???«, »AB«, »B*«, oder »*B« finden »ABC« dagegen nicht.

Zuletzt noch ein wichtiger Hinweis: Beim Compilieren der Find-Funktion mit SBasic muß die Integer-Variable »n%« immer auf 1 gesetzt sein, weil SBasic seine Zeichenketten völlig anders als GW-Basic organisiert und deshalb beim Aufruf eines Maschinencode-Unterprogramms mit »call« nur einzelne String-Variablen (und keine Felder) verwalten kann.

(Wolfgang Lorenz/ma)

Name	find.bas
Sprache	GW-Basic, SBasic
Grafikkarte	jede
Besonderheiten	keine


```
[3MA] 10 MC2$=SPACES(156):FOR I%=1 TO 77:READ X%:MID$(M
C2$,I%*2+1,2)=MKI$(X%):NEXT
[YL] 11 MC3$=SPACES(184):FOR I%=0 TO 91:READ X%:MID$(M
C3$,I%*2+1,2)=MKI$(X%):NEXT
[5S8] 12 CONVERT=PEEK(VARPTR(MC2$)+1)+PEEK(VARPTR(MC2$
+2)*256+2
[3C3] 13 BORDERS=CONVERT+80
[EZ7] 14 FIND=PEEK(VARPTR(MC3$)+1)+PEEK(VARPTR(MC3$)+2)
*256
[WM4] 15 MID$(MC2$,1,2)=MKI$(FIND-65450!)
[DSA] 100 DATA &HEC8B,&H8BFC,&H0676,&H8AAC,&HB5C8,&HE30
0,&HAD40,&HF08B,&H5E8B,&H8B04
[H7A] 101 DATA &H3D07,&H0001,&H2674,&H3177,&H3CAC,&H724
1,&H3C18,&H775A,&H0402,&H3C20
[1DA] 102 DATA &H758E,&HB002,&H3C84,&H7599,&HB002,&H3C9
4,&H759A,&HB002,&H8881,&HFF44
[5NA] 103 DATA &HDEE2,&H0DEB,&HEBAC,&H0000,&HFF5B,&HBA5
7,&H4488,&HE2FF,&HCAF3,&H0004
[7RA] 104 DATA &H8BFC,&H8BEC,&H0676,&H1C8B,&H8B4B,&H047
6,&H0C8B,&H768B,&HAC08,&HD08A
[5PA] 105 DATA &H00B6,&H348B,&HFF33,&HD32B,&H1E72,&HF30
3,&HCA3B,&H0276,&HCA8B,&H3CAC
[NHA] 106 DATA &H7520,&H4305,&HF8E2,&H0CEB,&HF103,&H4E4
E,&H3C80,&HE120,&H8BFA,&H47F9
[ZHA] 107 DATA &H768B,&H4306,&H1C89,&H768B,&H8904,&HCA3
C,&H0006,&HEC8B,&H5E8B,&H330A
[PKA] 108 DATA &H8BC9,&H0876,&H0C3B,&H3D73,&H768B,&H8A0
C,&H8B14,&H0174,&H378A,&H8B43
[GHA] 109 DATA &H433F,&H5343,&H5E8B,&H8A06,&H8A3F,&H22D
A,&H74DB,&H800E,&H2A3C,&H0474
[WKA] 110 DATA &H00B3,&H11EB,&HFE46,&H75CA,&H220C,&H5BD
B,&H1275,&HF622,&H0E74,&HEB41
[WRA] 111 DATA &HE8C4,&H0030,&HDB72,&HEB5B,&HB9F5,&HFFF
F,&H5E8B,&H8904,&HCA0F,&H000A
[PDA] 112 DATA &H613C,&H0672,&H7A3C,&H0377,&H202C,&H3CC
3,&H7584,&HB003,&HC38E,&H943C
[SAA] 113 DATA &H0375,&H99B0,&H3CC3,&H7581,&HB002,&HC39
A,&H5652,&H2257,&H74D2,&H8005
```

```
[BHA] 114 DATA &H2A3C,&H0575,&HC483,&HF906,&H22C3,&H75F
6,&H8305,&H06C4,&HC3F8,&H8AAC
[DMA] 115 DATA &H4725,&HEA81,&H0101,&H3F3C,&HDD74,&HFF2
2,&H0874,&HB3E8,&H86FF,&HE8C4
[B4A] 116 DATA &HFFAE,&HC43A,&HCD74,&HDB22,&HDB74,&H5E5
F,&H475A,&HCEFE,&HBEEB
[EE7] 200 '*** Demonstration für die Find-Funktion ***
[G22] 210 DEFINT A-Z
[G41] 220 CLS
[BL2] 230 DIM V$(40),N$(40)
[KC2] 240 FOR I=0 TO 11
[GN2] 250 READ V$(I),N$(I)
[PT3] 260 PRINT " ";V$(I);" ";N$(I)
[881] 270 NEXT
[MY7] 280 DATA Isaac,Asimov,John,Brunner,Arthur,Clarke
[MS8] 290 DATA Philip,Dick,William,Gibson,Robert,Heinle
in
[UC8] 300 DATA Frank,Herbert,Stanislaw,Lem,George,Orwel
l
[4T8] 310 DATA Frederik,Pohl,Robert,Silverberg,Jules,Ve
rne
[A42] 320 LOCATE 16,1
[R45] 330 LINE INPUT "Suchstring: ",S$
[9U2] 340 IF S$="" THEN END
[AY1] 350 LOCATE 1,1
[UC2] 360 FOR I=0 TO 11
[201] 370 N=1
[301] 380 M=1
[BG4] 390 CALL FIND!(S$,V$(I),N,M,EV)
[UD4] 400 CALL FIND!(S$,N$(I),N,M,EN)
[CZ6] 410 IF EV AND EN THEN PRINT " " ELSE PRINT "->"
[X71] 420 NEXT
[CP1] 430 GOTO 320
```

Listing. Die Basic-Erweiterung »find.bas« stellt Ihnen drei neue, leistungsfähige String-Funktionen zur Verfügung

Ordnung und Übersicht schaffen die beliebten DMV Sammelmappen



Bitte Bestellkarte benutzen
DMV Verlag · Postfach 250 · 3440 Eschwege

DMV informiert

Wir möchten Sie darauf hinweisen, daß sich unsere Telefonnummer geändert hat!

Die Telefonzentrale erreichen Sie unter:
0 56 51/80 09-0

Telefax:
0 56 51/80 09-33

DMV-Verlag
Postfach 250
3440 Eschwege

Profi-Tools für QuickBASIC und Turbo BASIC

Schreiben Sie schnellere, leistungsfähigere und professionellere Programme! Wir helfen Ihnen dabei mit nützlichen Tools.

Zum Beispiel:

- Toolboxes (Fenstertechnik, Menüs, DOS-Funktionen etc.)
- Relationale Datenbank mit komfortablem Masken-Editor
- Grafik-Paket (Geschäftsgrafik und Zeichensatz-Generator)
- Maus-Programmierung
- Laserdrucker-Unterstützung

Alle Pakete mit ausführlich dokumentierten Quelltexten und Programmbeispielen. Wo erforderlich, kommen schnelle Assembler-Routinen zum Einsatz. Wollen Sie mehr aus Ihrem BASIC-Compiler heraus holen? Wir informieren Sie gerne kostenlos!

Ingenieur-Büro
Harald Zoschke

Berliner Str. 3
D-2306 Schönberg/Holstein
Telefon 043 44/61 66

Eingetr. Warenzeichen:
QuickBASIC: Microsoft; Turbo BASIC: Borland

Mit »Parat« alle Daten am Start

Wozu benötigen Sie ein Adreßbuch, eine Karteikartensammlung oder ein Telefonnummernverzeichnis, wenn Sie doch einen PC besitzen? Das universelle Datenbankprogramm »Parat« räumt auf mit der Zettelwirtschaft und schafft Platz für wichtigere Dinge.

Sie haben vor, Adressen zu verwalten, Schallplatten zu sammeln, Bücher zu archivieren oder Videokassetten zu ordnen? Dann ist »Parat« genau das richtige für Sie, denn Parat ist ein Datenbankprogramm, das sehr flexibel ist und Ihnen viel Arbeit abnehmen kann. So hängt beispielsweise die Größe beziehungsweise die maximale Satzanzahl einer Datenbank, die Parat bearbeiten kann, ausschließlich vom zur Verfügung stehenden Arbeitsspeicher ab. Auch können Sie die Satzanzahl selbst begrenzen, um den restlichen Speicherplatz für zusätzlich Anwendungen zu nutzen.

Parat ist in GW-Basic geschrieben und läßt sich problemlos mit den Compilern Turbo Basic, Quick Basic und mit wenigen Änderungen auch mit SBasic (siehe Textkasten) übersetzen. Auf diese Weise kommen Sie in den Genuß höherer Ausführungsgeschwindigkeiten und größerer Datenbanken, denn die Basic-Compiler können mehr Daten verwalten als GW-Basic. Wenn allerdings in diesem Zusammenhang eine Fehlermeldung in der Art »out of memory« oder »out of string space« auftritt, müssen Sie die Sicherheitsreserve von 4096 Byte in Zeile 2620 so weit erhöhen, bis die Fehlermeldung nicht mehr auftritt.

Parat ist komplett menügeführt und besteht aus den folgenden Programmteilen:

Hauptmenü

Im Hauptmenü von Parat (Bild 1) erreichen Sie folgende Unterfunktionen:

- Dateidefinition
- Eingabe der Daten
- Datei pflegen
- Ausgabe der Daten
- Datei speichern
- Datei laden
- Beenden

Dateidefinition:

Bevor Sie eine neue Dateistruktur definieren, überprüft Parat, ob sich schon eine Datei im Arbeitsspeicher befindet. Ist das der Fall, dann erhalten Sie eine Fehlermeldung, und das Programm kehrt ins Hauptmenü zurück. Andernfalls legen Sie die Form einer Datei fest, indem Sie Parat zuerst die Anzahl der Felder mitteilen, die pro Datensatz abgespeichert werden sollen (Bild 2).

Zu diesem Zweck verfügt Parat über ein Unterprogramm, das die unkomfortable und schwer kontrollierbare Input-Anweisung ersetzt. So lassen sich beispielsweise nur Zahlen eingeben, wenn numerische Werte erwartet werden. Jede Eingabe können Sie durch Drücken der Esc-Taste abbrechen. Zusätzlich begrenzt Parat automatisch die Länge eines Datenfeldes. Aus diesen Gründen ist das Datenbankprogramm sehr bedienungsfreundlich; eine Fehlbedienung ist weitgehend ausgeschlossen.

Nachdem Sie die Anzahl der Felder pro Datensatz festgelegt haben, bestimmen Sie die Feldbezeichnung, den Feldtyp (alphanumerisch oder numerisch) und die Feldlänge. Stellen Sie anschließend fest, daß die Angaben nicht korrekt gewesen sind, erhalten Sie Gelegenheit, den Datensatz zu korrigieren. Wenn alle Angaben zutreffen, errechnet Parat anhand des unter GW-Basic zur Verfügung stehenden Arbeitsspeichers die maximale Anzahl der Datensätze und gibt sie auf dem Bildschirm aus.

Falls Sie Speicherplatz für zusätzliche Anwendungen reservieren wollen, ändern Sie die Datensatzobergrenze einfach wie gewünscht ab.

Eingabe der Daten:

Jeder Datensatz wird Feld für Feld eingegeben. Parat zeigt Ihnen die Nummer des aktuellen Satzes und die maximal mögliche Satznummer. Auf dem Bildschirm erscheinen die vordefinierten Feldbezeichnungen, und Sie werden aufgefordert, die zugehörigen Daten einzutippen. Stellen Sie nach der Eingabe fest, daß Sie sich bei der Eingabe geirrt haben, so können Sie diese verbessern. Andernfalls fahren Sie mit dem nächsten Datensatz fort. Parat erhöht die Satznummer automatisch um den Wert Eins, bis die maximale Satzanzahl erreicht ist.

Dateibearbeitung:

In diesem Untermenü (Bild 3) erreichen Sie folgende Unterfunktionen:

- Dateidefinition löschen
- Datensatz ändern
- Datensatz löschen
- Datei sortieren
- Hauptmenü

Dateidefinition löschen:

Diese Funktion löscht die im Arbeitsspeicher vorhandene Dateidefinition.

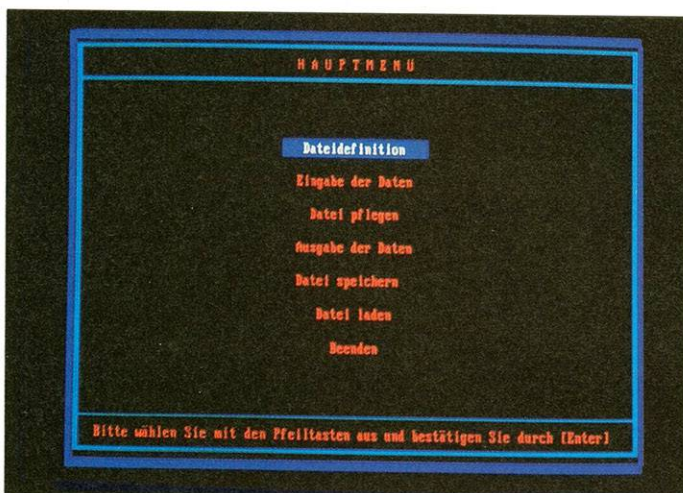


Bild 1. Im Hauptmenü von »Parat« sind die wichtigsten Programmfunktionen zugänglich

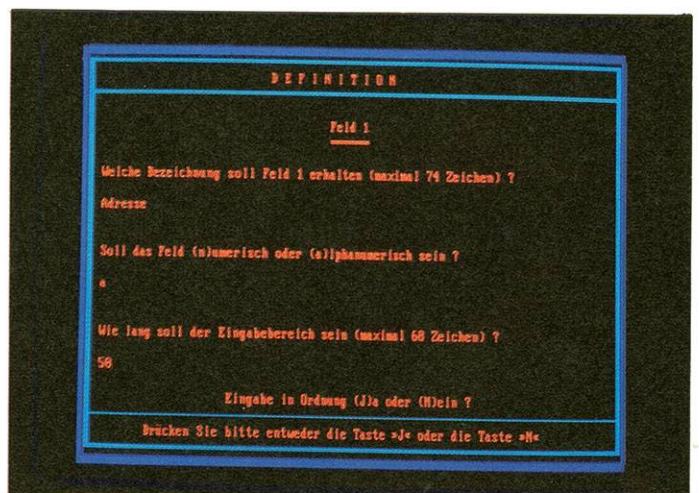


Bild 2. In dem Untermenü »Dateidefinition« legen Sie das Format der Datensätze fest

Datensatz ändern:

Nachdem Sie die Nummer des Datensatzes eingegeben haben, zeigt Ihnen Parat die zugehörigen Datensatzfelder an.

Datensatz löschen:

Nach der Eingabe der Nummer des gewünschten Datensatzes zeigt Ihnen Parat dessen Inhalt an. Daraufhin bestimmen Sie, ob der Datensatz gelöscht werden soll oder nicht.

Datei sortieren:

Die Datei wird unter Berücksichtigung des angegebenen Schlüsselfeldes sortiert.

Hauptmenü:

Wenn Sie diesen Menüpunkt anwählen, kehrt Parat in das Hauptmenü zurück.

Ausgabe:

Parat verfügt über vielfältige Ausgabemöglichkeiten, die in dem Ausgabemenü zum Ausdruck kommen:

- Einzelne Sätze
- Ausgewählte Sätze
- Alle Sätze
- Hauptmenü

Einzelne Sätze:

Parat gibt nach der Eingabe der Satznummer den gewünschten Datensatz auf dem Bildschirm und auf Wunsch zusätzlich auf einem Drucker aus. Mit den Tasten [V] und [Z] blättern Sie jeweils einen Datensatz vor und zurück.

Ausgewählte Sätze:

Dieser Menüpunkt gehört zu den leistungsstärksten Funktionen von Parat. Das Datenbankprogramm erwartet zuerst die Angabe eines Suchschlüssels und darauf das Suchkriterium. Parat stellt Ihnen die drei Vergleichsoperatoren »>«, »<« und »=« zur Verfügung, um diese für vergleichende Suchoperationen einsetzen zu können. Der Platzhalter »#«

So compilieren Sie »Parat« mit SBasic

Um »Parat« mit SBasic erfolgreich compilieren zu können, müssen Sie folgende Änderungen vornehmen:

Ersetzen Sie zuerst die Programmzeile 1040 durch

```
1040 view print 1 to 25
```

Entfernen Sie darauf die Programmzeilen 2620 bis 2700 sowie 6900 und 7190 ganz aus dem Listing.

Weil SBasic dynamische Dimensionierungen nicht zuläßt, müssen Sie die Felder in Zeile 2270 und 2710 statisch, das heißt mit einem festen Wert anstelle einer Variablen dimensionieren.

Für die Variable »felder« in Zeile 2270 und 2710 setzen Sie die Anzahl der gewünschten Felder pro Datensatz und für die Variable »grenze« in Zeile 2710 die Ihrer Meinung nach maximale Anzahl der Datensätze ein. Ebenso ersetzen Sie die Programmzeile 2610 durch eine feste Zuweisung für die Variable »grenze«.

Möchten Sie beispielsweise eine Datei mit fünf Feldern pro Datensatz und maximal 100 Datensätzen anlegen, dann müssen Sie die Zeilen 2270, 2610 und 2710 wie folgt ändern:

```
2270 dim index$(5), laenge(5), art$(5)
```

```
2610 grenze = 100
```

```
2710 dim satz$(100,5)
```

Die maximale Anzahl der Datensätze ist übrigens von der Gesamtlänge aller Datensätze und der Größe Ihres Arbeitsspeicher abhängig. Weil SBasic allerdings wesentlich mehr Arbeitsspeicher als GW-Basic verwalten kann, dürfen Sie für »grenze« auch größere Werte einzusetzen, als es Ihnen GW-Basic erlaubt. Den maximal zulässigen Wert müssen Sie allerdings selbst durch Ausprobieren ermitteln.

Wenn Sie ganz auf Nummer sicher gehen wollen, setzen Sie nur den Wert ein, den Ihnen die GW-Basic-Version vorschlägt. Überschreiten Sie bei der Dateidefinition oder der Dateneingabe allerdings die Werte für »felder« und »grenze«, so gehen Ihnen diese Daten verloren, weil sie nicht korrekt abgelegt werden.

entspricht in bezug auf die Funktionsweise dem MS-DOS-Wildcard »*«. Stellen Sie dem Suchkriterium kein Vergleichszeichen voran, dann setzt Parat automatisch das Gleichheitszeichen vor die Zeichenkette. Verliefe die Suche erfolgreich, dann gibt Parat alle gefundenen Datensätze auf dem Bildschirm und auf Wunsch auch auf den Drucker aus. Die Beispiele in der Tabelle verdeutlichen Ihnen das Funktionsprinzip der Suchroutine von Parat.

Alle Sätze:

Parat gibt alle im Arbeitsspeicher enthaltenen Datensätze ab Satznummer 1 auf dem Bildschirm und auf Wunsch zusätzlich auf einen Drucker

aus. Sie blättern mit den Tasten »V« und »Z« jeweils einen Datensatz vor und zurück.

Datei speichern:

Nach der Eingabe des Dateinamens speichert Parat die Systemparameter und Datensätze auf Diskette oder Festplatte ab.

Datei laden:

Parat liest nach der Angabe eines Dateinamens die zugehörige Dateidefinition und die einzelnen Datensätze vom gewünschten Laufwerk.

Im Listing von Parat finden Sie einige nützliche Programmiertricks, die Sie auch in anderen Programmen verwenden können. So ist Parat in mehrere Unterprogramme aufgeteilt, die in fast allen Fällen jeweils für ein Untermenü verantwortlich sind.

In dem Programmkopf ab Zeile 1000 initialisiert Parat alle Variablen und gibt das Titelbild auf dem Bildschirm aus. In Zeile 1460 beginnt das Haupt-

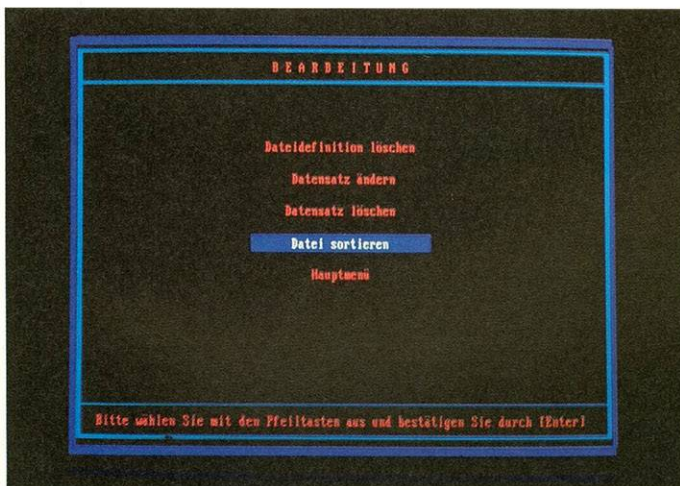


Bild 3. Das Untermenü »Bearbeitung« erlaubt unter anderem das Sortieren der Daten

Suchbegriff	Gefundener Begriff
4322	Alle Datensätze, deren Feldschlüssel gleich dem Wert 4322 ist
T #	Alle Datensätze, deren Feldschlüssel mit einem großen »T« beginnt
# chen #	Alle Datensätze, deren Feldschlüssel ein »chen« enthält
> 1	Alle Datensätze, deren Feldschlüssel größer als 1 ist
< Mün	Alle Datensätze, deren Feldschlüssel kleiner als »Mün« ist

Tabelle. Die Suchroutine des Datenbankprogramms »Parat« ist universell einsetzbar

menü. Die Prozedur ab Zeile 1740 fängt Bedienungsfehler ab und gibt die Parat-Fehlermeldungen an den Benutzer weiter. Das Unterprogramm in dem Bereich von Zeile 1830 bis 2060, dessen Einsatz auch in anderen Programmen nützlich ist, übernimmt die Anwendereingaben. Zu diesem Zweck übergeben Sie in der Variablen »laenge« die Länge der einzugebenden Zeichenkette sowie in »x« und »y« die Anfangsposition.

Die Variable »menge\$« bestimmt die erlaubten Zeichen. Enthält »menge\$« die Angabe »zahl«, dann werden bei der Dateneingabe nur Zahlen berücksichtigt. Ist in »menge\$« hingegen das Wort »alle« gespeichert, sind alle Zeichen erlaubt. Andere Werte beschränken die erlaubten Zeichen auf die angegebenen Zeichen.

Wie bei vielen Basic-Programmen so gibt es auch bei Parat ein Problem, das Kopfzerbrechen bereitet: Varia-

blen-Felder müssen mit dem Befehl »clear« gelöscht werden; dabei löscht der Interpreter aber auch den Inhalt von Variablen, die noch benötigt werden.

Es gibt zwei Alternativen, um die Daten zwischenspeichern, die später noch gebraucht werden. Beide Methoden finden bei Parat Verwendung. Die erste Methode finden Sie ab Zeile 2230. Parat gibt die Anzahl der zu definierenden Datenfelder auf dem Bildschirm aus und liest sie nach Ausführung des Clear-Befehls mit der Screen-Funktion wieder in die zugehörige Variable ein. Das zweite Verfahren gestaltet sich etwas aufwendiger. Dabei speichert Parat die Positionen innerhalb der einzelnen Menüs (wie beispielsweise in dem Unterpro-

gramm ab Zeile 6690) in der Datei »\$. \$« ab.

Dann führt Parat den Clear-Befehl aus und liest die Daten wieder aus der Datei in den Arbeitsspeicher zurück. Interessant ist auch das Unterprogramm in dem Bereich von Zeile 4190 bis 4610. Diese Programmzeilen sortieren die eingegebenen Daten unter Berücksichtigung des in der Variablen »satz\$« angegebenen Sortierkriteriums. Parat sortiert mit Hilfe des schnellen Quick-Sort-Algorithmus, der die zu sortierenden Daten in immer kleinere Mengen einteilt.

(Markus Zietlow/ma)

Name	parat.bas
Sprache	GW-Basic, SBasic
Grafikkarte	jede
Besonderheiten	keine

```
[DK2] 1000 *****
[GT3] 1010 * Initialisierung *
[NK2] 1020 *****
[3N0] 1030 '
[BS1] 1040 KEY OFF
[241] 1050 CLS
[8VA] 1060 HAUPTMENUE = 7: BEARBEITUNG = 7: DATENAUSGAB
E = 7: AUSDRUCKEN = 7
[DA2] 1070 COLOR 3, 0, 1
[YY2] 1080 PARAT$ = "PARAT "
[0U6] 1090 PRINT CHR$(201); STRING$(78, 205); CHR$(187)
[C4D] 1100 FOR Y = 2 TO 24: LOCATE Y, 1: PRINT CHR$(186)
); : LOCATE Y, 80: PRINT CHR$(186); : NEXT Y
[BU8] 1110 LOCATE 25, 1: PRINT CHR$(200); STRING$(78, 2
05); CHR$(188);
[PS8] 1120 LOCATE 3, 1: PRINT CHR$(204); STRING$(78, 20
5); CHR$(185);
[FV8] 1130 LOCATE 23, 1: PRINT CHR$(199); STRING$(78, 1
96); CHR$(182);
[PZ3] 1140 FOR SCHLEIFE = 4 TO 22
[WE3] 1150 ZEIGER = ZEIGER + 1
[X95] 1160 IF ZEIGER = 7 THEN ZEIGER = 1
[WG3] 1170 LOCATE SCHLEIFE, 2
[X04] 1180 FOR SCHLEIFE = 1 TO 13
[XX8] 1190 PRINT MID$(PARAT$, ZEIGER); LEFT$(PARAT$,
ZEIGER - 1);
[103] 1200 NEXT SCHLEIFE2
[5U2] 1210 NEXT SCHLEIFE
[7P1] 1220 COLOR 2
[0C6] 1230 LOCATE 2, 28: PRINT "P A R A
T"
[GR8] 1240 LOCATE 5, 24: PRINT CHR$(218); STRING$(31, 1
96); CHR$(191)
[M6C] 1250 LOCATE 6, 24: PRINT CHR$(179); "Die universe
lle Dateiverwaltung"; CHR$(179)
[RR8] 1260 LOCATE 7, 24: PRINT CHR$(192); STRING$(31, 1
96); CHR$(217)
[5H8] 1270 LOCATE 9, 38: PRINT CHR$(218); STRING$(3, 19
6); CHR$(191)
[1F7] 1280 LOCATE 10, 38: PRINT CHR$(179); "von"; CHR$(
179)
[XV8] 1290 LOCATE 11, 38: PRINT CHR$(192); STRING$(3, 1
96); CHR$(217)
[2X8] 1300 LOCATE 13, 33: PRINT CHR$(218); STRING$(14,
196); CHR$(191)
[XB9] 1310 LOCATE 14, 33: PRINT CHR$(179); "Markus Ziet
low"; CHR$(179)
[FX8] 1320 LOCATE 15, 33: PRINT CHR$(192); STRING$(14,
196); CHR$(217)
[KY8] 1330 LOCATE 19, 11: PRINT CHR$(218); STRING$(58,
196); CHR$(191)
[5AG] 1340 LOCATE 20, 11: PRINT CHR$(179); "(C) 1989 Re
daktionsbüro Everts&Hagedorn, DOS International";
CHR$(179)
[T19] 1350 LOCATE 21, 11: PRINT CHR$(192); STRING$(58,
196); CHR$(217);
[XT1] 1360 COLOR 12
[359] 1370 MELDUNG$ = "Drücken Sie bitte eine beliebige
```

```
Taste. "
[T82] 1380 ZEIGER = 1
[X13] 1390 WHILE INKEY$ = ""
[VK2] 1400 LOCATE 24, 22
[PM9] 1410 PRINT MID$(MELDUNG$, ZEIGER); LEFT$(MELDUNG
$, ZEIGER - 1);
[TE3] 1420 ZEIGER = ZEIGER + 1
[YM7] 1430 IF ZEIGER = LEN(MELDUNG$) + 1 THEN ZEIGER =
1
[NC6] 1440 FOR SCHLEIFE = 1 TO 100: NEXT SCHLEIFE
[H91] 1450 WEND
[VN0] 1460 '
[L22] 1470 *****
[FX2] 1480 * Hauptmenü *
[V22] 1490 *****
[6N0] 1500 '
[GP1] 1510 COLOR 6
[ZG5] 1520 GOSUB 7230: GOSUB 7290: GOSUB 7350
[U13] 1530 Y = HAUPTMENUE
[YH6] 1540 LOCATE 2, 32: PRINT "H A U P T M E N Ü"
[NC6] 1550 LOCATE 7, 33: PRINT "Dateidefinition"
[LG6] 1560 LOCATE 9, 32: PRINT "Eingabe der Daten"
[146] 1570 LOCATE 11, 34: PRINT "Datei pflegen"
[DM6] 1580 LOCATE 13, 32: PRINT "Ausgabe der Daten"
[AM6] 1590 LOCATE 15, 32: PRINT "Datei speichern"
[CUS] 1600 LOCATE 17, 35: PRINT "Datei laden"
[4A5] 1610 LOCATE 19, 37: PRINT "Beenden "
[C6G] 1620 LOCATE 24, 4: PRINT "Bitte wählen Sie mit de
n Pfeiltasten aus und bestätigen Sie durch [Enter
]";
[BED] 1630 COLOR 14, 1: FOR Q = 30 TO 50: LOCATE Y, Q:
PRINT CHR$(SCREEN(Y, Q)); NEXT Q: COLOR 6, 0
[2G6] 1640 TASTE$ = INKEY$: IF TASTE$ = "" THEN 1640
[D76] 1650 IF TASTE$ <> CHR$(0) + CHR$(72) THEN 1680
[0GD] 1660 IF Y <> 7 THEN FOR Q = 30 TO 50: LOCATE Y, Q
: PRINT CHR$(SCREEN(Y, Q)); NEXT Q: Y = Y - 2
[3X1] 1670 GOTO 1630
[776] 1680 IF TASTE$ <> CHR$(0) + CHR$(80) THEN 1710
[7TD] 1690 IF Y <> 19 THEN FOR Q = 30 TO 50: LOCATE Y,
Q: PRINT CHR$(SCREEN(Y, Q)); NEXT Q: Y = Y + 2
[AW1] 1700 GOTO 1630
[X38] 1710 IF TASTE$ <> CHR$(13) THEN 1640 ELSE HAUPTME
NUE = Y
[5H8] 1720 ON (Y - 1) / 2 - 2 GOSUB 2090, 2760, 3110, 4
640, 6390, 6710, 7080
[YW1] 1730 GOTO 1480
[WN0] 1740 '
[MR2] 1750 *****
[3Y3] 1760 * Fehlerbehandlung *
[WR2] 1770 *****
[CP0] 1780 '
[XB5] 1790 LOCATE Y, (41 - (LEN(FEHLER$))) / 2
[4GA] 1800 PRINT FEHLER$; " - Drücken Sie bitte eine be
liebige Taste. ";
[D14] 1810 WHILE INKEY$ = "": WEND
[SV1] 1820 RETURN
[VN0] 1830 '
[H54] 1840 *****
```



```

[AC6] 1850 '* Unterprogramm zur Dateneingabe *
[T54] 1860 '*****
[BP0] 1870 '
[1C1] 1880 QS = ""
[1C2] 1890 LOCATE Y, X
[VC3] 1900 PRINT QS; CHR$(219);
[WF6] 1910 TASTE$ = INKEY$: IF TASTE$ = "" THEN 1910
[N15] 1920 IF (TASTE$ <> CHR$(8)) THEN 1980
[R24] 1930 IF LEN(QS) = 0 THEN 1910
[1A4] 1940 QS = LEFT$(QS, LEN(QS) - 1)
[YF2] 1950 LOCATE Y, X
[214] 1960 PRINT QS; CHR$(219); " ";
[R32] 1970 GOTO 1910
[0T8] 1980 IF (TASTE$ = CHR$(13)) OR (TASTE$ = CHR$(27))
    THEN RETURN
[VZ4] 1990 IF (ASC(TASTE$) < 32) THEN 1910
[6P4] 2000 IF MENGE$ <> "zahl" THEN 2030
[318] 2010 IF (ASC(TASTE$) < 48) OR (ASC(TASTE$) > 57)
    THEN 1910
[122] 2020 GOTO 2050
[GF4] 2030 IF MENGE$ = "alle" THEN 2050
[CE6] 2040 IF INSTR(MENGE$, TASTE$) = 0 THEN 1910
[4K6] 2050 IF LEN(QS) < LAENGE THEN QS = QS + TASTE$
[XW1] 2060 GOTO 1890
[NN0] 2070 '
[DR2] 2080 '*****
[MT3] 2090 '* Datei definieren *
[GP2] 2100 '*****
[0N0] 2110 '
[NG5] 2120 GOSUB 7230: GOSUB 7290: GOSUB 7350
[FE6] 2130 LOCATE 2, 28: PRINT "D E F I N I T I O N"
[NKD] 2140 IF AKTUELL > 0 THEN Y = 10: FEHLER$ = "Datei
    bereits definiert": GOSUB 1760: RETURN
[VAD] 2150 LOCATE 10, 10: PRINT "Wie viele Datenfelder
    wollen Sie definieren (maximal 17) ?"
[B78] 2160 Y = 10: X = 69: LAENGE = 2: MENGE$ = "zahl":
    GOSUB 1880
[NT5] 2170 IF (TASTE$ = CHR$(27)) THEN RETURN
[MHA] 2180 IF (QS = "") OR (VAL(QS) = 0) OR (VAL(QS) >
    17) THEN GOSUB 1890: GOTO 2170
[1K4] 2190 LOCATE Y, X: PRINT QS; " ";
[S23] 2200 FELDER = VAL(QS)
[A4B] 2210 LOCATE 24, 22: PRINT "Anzahl der zu definier
    enden Felder:"; FELDER;
[AZZ] 2220 OPEN "$$. " FOR OUTPUT AS #2: WRITE #2, HAUPT
    MENUE, BEARBEITUNG, DATENAUSGABE, AUSDRUCKEN: CLO
    SE : CLEAR : OPEN "$$. " FOR INPUT AS #2: INPUT #2
    , HAUPTMENUE, BEARBEITUNG, DATENAUSGABE, AUSDRUCK
    EN: CLOSE
[2D1] 2230 P = 58
[7R5] 2240 IF SCREEN(24, P) = 32 THEN GOTO 2270
[9G7] 2250 FELDER = FELDER * 10 + SCREEN(24, P) - 48: P
    = P + 1
[RW1] 2260 GOTO 2240
[M48] 2270 DIM INDEX$(FELDER), LAENGE(FELDER), ART$(FEL
    DER)
[8E2] 2280 SCHLEIFE = 1
[Y42] 2290 GOSUB 7350
[94B] 2300 LOCATE 24, 22: PRINT "Anzahl der zu definier
    enden Felder:"; FELDER;
[B42] 2310 GOSUB 7290
[MZ4] 2320 FELD$ = "Feld" + STR$(SCHLEIFE)
[2U6] 2330 LOCATE 5, (80 - LEN(FELD$)) / 2: PRINT FELD$
[7N9] 2340 LOCATE 6, (80 - LEN(FELD$)) / 2: PRINT STRIN
    G$(LEN(FELD$), 205)
[TPF] 2350 LOCATE 8, 3: PRINT "Welche Bezeichnung soll
    Feld"; SCHLEIFE; "erhalten (maximal 74 Zeichen) ?"
[R48] 2360 Y = 10: X = 3: LAENGE = 74: MENGE$ = "alle":
    GOSUB 1880
[0V5] 2370 IF (TASTE$ = CHR$(27)) THEN GOTO 1480
[3K4] 2380 LOCATE Y, X: PRINT QS; " ";
[3V3] 2390 INDEX$(SCHLEIFE) = QS
[VFC] 2400 LOCATE 13, 3: PRINT "Soll das Feld (n)umeris
    ch oder (a)lphanumerisch sein ?"
[0Z7] 2410 Y = 15: X = 3: LAENGE = 1: MENGE$ = "aAnN":
    GOSUB 1880
[FU5] 2420 IF (TASTE$ = CHR$(27)) THEN GOTO 1480
[RN5] 2430 IF QS = "" THEN GOSUB 1890: GOTO 2420
[PH4] 2440 LOCATE Y, X: PRINT QS; " ";
[6K6] 2450 IF ASC(QS) > 90 THEN QS = CHR$(ASC(QS) - 32)
[8H3] 2460 ART$(SCHLEIFE) = QS
[7MH] 2470 LOCATE 18, 3: PRINT "Wie lang soll der Eing
    abereich sein (maximal); 75 - LEN(INDEX$(SCHLEIF
    E)); "Zeichen" ?"
[0Z8] 2480 Y = 20: X = 3: LAENGE = 2: MENGE$ = "zahl":
    GOSUB 1880
[BV5] 2490 IF (TASTE$ = CHR$(27)) THEN GOTO 1480
[V7E] 2500 IF (QS = "") OR (VAL(QS) < 1) OR (VAL(QS) >
    75 - LEN(INDEX$(SCHLEIFE))) THEN GOSUB 1890: GOTO
    2490
[DH4] 2510 LOCATE Y, X: PRINT QS; " ";
[HR4] 2520 LAENGE(SCHLEIFE) = VAL(QS)

```

```

[DD9] 2530 LOCATE 22, 22: PRINT "Eingabe in Ordnung (J)
    a oder (N)ein ?"
[AVE] 2540 LOCATE 24, 10: PRINT "Drücken Sie bitte entw
    eder die Taste »J« oder die Taste »N«";
[EC7] 2550 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 25
    50
[T57] 2560 IF INSTR("jJnN", TASTE$) = 0 THEN GOTO 2550
[PH6] 2570 IF INSTR("jJ", TASTE$) = 0 THEN GOTO 2290
[485] 2580 IF SCHLEIFE = FELDER THEN 2610
[244] 2590 SCHLEIFE = SCHLEIFE + 1
[NW1] 2600 GOTO 2290
[TAB] 2610 FOR SCHLEIFE = 1 TO FELDER: GESAMT = GESAMT
    + LAENGE(SCHLEIFE): NEXT
[VX7] 2620 GRENZE = INT((FRE("") - 4096) / (GESAMT + FE
    LDER * 10))
[6V3] 2630 GOSUB 7350: GOSUB 7290
[12C] 2640 LOCATE 10, (80 - (20 + LEN(STR$(GRENZE)))) /
    2: PRINT "Maximale Satzanzahl:"
[ZSA] 2650 LOCATE 22, 20: PRINT "Geben Sie die gewünsch
    te Satzanzahl ein"
[A1M] 2660 Y = 10: X = (80 - (20 + LEN(STR$(GRENZE))))
    / 2 + 21: LAENGE = LEN(STR$(GRENZE)) - 1: MENGE$
    = "zahl": QS = MID$(STR$(GRENZE), 2): GOSUB 1890
[BV5] 2670 IF (TASTE$ = CHR$(27)) THEN GOTO 2660
[YFB] 2680 IF (QS = "") OR (VAL(QS) = 0) OR (VAL(QS) >
    GRENZE) THEN GOSUB 1890: GOTO 2670
[GK4] 2690 LOCATE Y, X: PRINT QS; " ";
[P53] 2700 GRENZE = VAL(QS)
[DE4] 2710 DIM SATZ$(GRENZE, FELDER)
[192] 2720 AKTUELL = 1
[0X1] 2730 GOTO 1480
[YN0] 2740 '
[GT1] 2750 '*****
[2A2] 2760 '* Eingabe *
[ST1] 2770 '*****
[EP0] 2780 '
[3K5] 2790 GOSUB 7230: GOSUB 7290: GOSUB 7350
[DH5] 2800 LOCATE 2, 34: PRINT "E I N G A B E"
[TXE] 2810 IF AKTUELL = 0 THEN Y = 10: FEHLER$ = "Keine
    Datenstruktur definiert": GOSUB 1760: GOTO 3080
[CC5] 2820 IF AKTUELL <= GRENZE THEN 2840
[5G9] 2830 Y = 24: FEHLER$ = "Die Datei ist voll": GOSU
    B 1760: GOTO 3080
[GV3] 2840 GOSUB 7290: GOSUB 7350
[PUD] 2850 LOCATE 24, 16: PRINT "Maximale Satzanzahl:";
    GRENZE; "Aktueller Satz:"; AKTUELL;
[S71] 2860 Y = 4
[7W5] 2870 LOCATE Y, 3: PRINT INDEX$(Y - 3); " ";
[VL7] 2880 X = 4 + LEN(INDEX$(Y - 3)): LAENGE = LAENGE(
    Y - 3)
[SC9] 2890 IF ART$(Y - 3) = "A" THEN MENGE$ = "alle" EL
    SE MENGE$ = "zahl"
[W42] 2900 GOSUB 1880
[C26] 2910 IF (TASTE$ <> CHR$(27)) THEN GOTO 2960
[NV4] 2920 FOR SCHLEIFE = 1 TO FELDER
[B65] 2930 SATZ$(AKTUELL, SCHLEIFE) = ""
[8V2] 2940 NEXT SCHLEIFE
[KX1] 2950 GOTO 3080
[DK4] 2960 LOCATE Y, X: PRINT QS; " ";
[BB4] 2970 SATZ$(AKTUELL, Y - 3) = QS
[PF4] 2980 IF Y - 3 = FELDER THEN 3010
[MR1] 2990 Y = Y + 1
[7W1] 3000 GOTO 2870
[U32] 3010 GOSUB 7350
[LK9] 3020 LOCATE 24, 22: PRINT "Eingabe in Ordnung (J)
    a oder (N)ein ?";
[AB7] 3030 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 30
    30
[F47] 3040 IF INSTR("jJnN", TASTE$) = 0 THEN GOTO 3030
[TG6] 3050 IF INSTR("jJ", TASTE$) = 0 THEN GOTO 2820
[LU3] 3060 AKTUELL = AKTUELL + 1
[TW1] 3070 GOTO 2820
[WV1] 3080 RETURN
[ZN0] 3090 '
[KP2] 3100 '*****
[2X3] 3110 '* Dateibearbeitung *
[UP2] 3120 '*****
[AN0] 3130 '
[ZG5] 3140 GOSUB 7230: GOSUB 7290: GOSUB 7350
[2P6] 3150 LOCATE 2, 30: PRINT "B E A R B E I T U N G"
[BXD] 3160 IF AKTUELL = 0 THEN Y = 10: FEHLER$ = "Datei
    noch nicht definiert": GOSUB 1760: RETURN
[N23] 3170 Y = BEARBEITUNG
[X77] 3180 LOCATE 7, 29: PRINT "Dateidefinition löschen"
[AV6] 3190 LOCATE 9, 33: PRINT "Datensatz ändern"
[S57] 3200 LOCATE 11, 32: PRINT "Datensatz löschen"
[UL6] 3210 LOCATE 13, 33: PRINT "Datei sortieren"
[YM5] 3220 LOCATE 15, 36: PRINT "Hauptmenü"

```

Listing. »Parat« hält Ihre Daten zu jedem Zeitpunkt für Sie bereit


```
[86G] 3230 LOCATE 24, 4: PRINT "Bitte wählen Sie mit de
n Pfeiltasten aus und bestätigen Sie durch [Enter
]";
[LED] 3240 COLOR 14, 1: FOR Q = 27 TO 53: LOCATE Y, Q:
PRINT CHR$(SCREEN(Y, Q)): NEXT Q: COLOR 6, 0
[4G6] 3250 TASTE$ = INKEY$: IF TASTE$ = "" THEN 3250
[F76] 3260 IF TASTE$ <> CHR$(0) + CHR$(72) THEN 3290
[KGD] 3270 IF Y <> 7 THEN FOR Q = 27 TO 53: LOCATE Y, Q
: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y - 2
[5X1] 3280 GOTO 3240
[976] 3290 IF TASTE$ <> CHR$(0) + CHR$(80) THEN 3320
[DSO] 3300 IF Y <> 15 THEN FOR Q = 27 TO 53: LOCATE Y,
Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y + 2
[5X1] 3310 GOTO 3240
[P78] 3320 IF TASTE$ <> CHR$(13) THEN 3250 ELSE BEARBEI
TUNG = Y
[6X3] 3330 IF Y = 15 THEN RETURN
[FD6] 3340 ON (Y - 1) / 2 - 2 GOSUB 3380, 3520, 3850, 4
210
[NW1] 3350 GOTO 3110
[WN0] 3360
[5D3] 3370 *****
[CF5] 3380 * Dateidefinition löschen *
[6E3] 3390 *****
[7N0] 3400
[5G5] 3410 GOSUB 7230: GOSUB 7290: GOSUB 7350
[X5A] 3420 LOCATE 2, 18: PRINT "D A T E I D E F I N I T
I O N L Ö S C H E N"
[VDE] 3430 LOCATE 10, 8: PRINT "Wollen Sie die Dateidef
inition wirklich löschen (J)a oder (N)ein ?"
[9VE] 3440 LOCATE 24, 10: PRINT "Drücken Sie bitte entw
eder die Taste »J« oder die Taste »N«";
[CC7] 3450 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 34
50
[R57] 3460 IF INSTR("jJnJ", TASTE$) = 0 THEN GOTO 3450
[PK6] 3470 IF INSTR("nn", TASTE$) <> 0 THEN RETURN
[A00] 3480 OPEN "$." FOR OUTPUT AS #2: WRITE #2, HAUPT
MENUE, BEARBEITUNG, DATENAUSGABE, AUSDRUCKEN: CLO
SE : CLEAR : OPEN "$." FOR INPUT AS #2: INPUT #2
, HAUPTMENUE, BEARBEITUNG, DATENAUSGABE, AUSDRUCK
EN: CLOSE
[HX1] 3490 GOTO 1480
[AN0] 3500
[2R2] 3510 *****
[C44] 3520 * Datensatz ändern *
[AR2] 3530 *****
[UN0] 3540
[PH5] 3550 GOSUB 7230: GOSUB 7290: GOSUB 7350
[4BA] 3560 LOCATE 2, 17: PRINT "Ä N D E R N E I N E S
D A T E N S A T Z E S"
[PYC] 3570 LOCATE 10, 16: PRINT "Welchen Satz wollen Si
e ändern (maximal"; AKTUELL - 1; ") ?"
[WXE] 3580 IF AKTUELL = 1 THEN Y = 10: FEHLER$ = "Es is
t kein Datensatz definiert": GOSUB 1760: RETURN
[188] 3590 Y = 10: X = 65: LAENGE = 5: MENGE$ = "zahl":
GOSUB 1880
[AT5] 3600 IF (TASTE$ = CHR$(27)) THEN RETURN
[2VB] 3610 IF (QS = "") OR (VAL(QS) = 0) OR (VAL(QS) >
AKTUELL - 1) THEN GOSUB 1890: GOTO 3600
[PH4] 3620 LOCATE Y, X: PRINT QS; " ";
[PT2] 3630 SATZ = VAL(QS)
[CV3] 3640 GOSUB 7290: GOSUB 7350
[UAD] 3650 LOCATE 24, 16: PRINT "Maximale Satzanzahl:";
GRENZE; " Aktueller Satz:"; SATZ;
[M71] 3660 Y = 4
[WT8] 3670 LOCATE Y, 3: PRINT INDEX$(Y - 3); " "; SATZ$
(SATZ, Y - 3)
[73B] 3680 X = 4 + LEN(INDEX$(Y - 3)): LAENGE = LAENGE(
Y - 3): QS = SATZ$(SATZ, Y - 3)
[MC9] 3690 IF ART$(Y - 3) = "A" THEN MENGE$ = "alle" EL
SE MENGE$ = "zahl"
[V42] 3700 GOSUB 1890
[HT5] 3710 IF (TASTE$ = CHR$(27)) THEN RETURN
[TH4] 3720 LOCATE Y, X: PRINT QS; " ";
[MW3] 3730 SATZ$(SATZ, Y - 3) = QS
[FG4] 3740 IF Y - 3 = FELDER THEN 3770
[OR1] 3750 Y = Y + 1
[PX1] 3760 GOTO 3670
[752] 3770 GOSUB 7350
[OM9] 3780 LOCATE 24, 22: PRINT "Eingabe in Ordnung (J)
a oder (N)ein ?";
[RD7] 3790 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 37
90
[M57] 3800 IF INSTR("jJnJ", TASTE$) = 0 THEN GOTO 3790
[7H6] 3810 IF INSTR("jJ", TASTE$) = 0 THEN GOTO 3640
[5V1] 3820 RETURN
[2N0] 3830
[ZW2] 3840 *****
[H54] 3850 * Datensatz löschen *
[7X2] 3860 *****
[FP0] 3870
[4K5] 3880 GOSUB 7230: GOSUB 7290: GOSUB 7350
[BUA] 3890 LOCATE 2, 16: PRINT "L Ö S C H E N E I N E
S D A T E N S A T Z E S"
[8XE] 3900 IF AKTUELL = 1 THEN Y = 10: FEHLER$ = "Es is
t kein Datensatz definiert": GOSUB 1760: RETURN
```

```
[YBD] 3910 LOCATE 10, 16: PRINT "Welchen Satz wollen Si
e löschen (maximal"; AKTUELL - 1; ") ?"
[H78] 3920 Y = 10: X = 65: LAENGE = 5: MENGE$ = "zahl":
GOSUB 1880
[2T5] 3930 IF (TASTE$ = CHR$(27)) THEN RETURN
[CWB] 3940 IF (QS = "") OR (VAL(QS) = 0) OR (VAL(QS) >
AKTUELL - 1) THEN GOSUB 1890: GOTO 3930
[BK4] 3950 LOCATE Y, X: PRINT QS; " ";
[BU2] 3960 SATZ = VAL(QS)
[1W3] 3970 GOSUB 7290: GOSUB 7350
[FBD] 3980 LOCATE 24, 16: PRINT "Maximale Satzanzahl:";
GRENZE; " Aktueller Satz:"; SATZ;
[981] 3990 Y = 4
[GS8] 4000 LOCATE Y, 3: PRINT INDEX$(Y - 3); " "; SATZ$
(SATZ, Y - 3)
[HE4] 4010 IF Y - 3 = FELDER THEN 4040
[1P1] 4020 Y = Y + 1
[7W1] 4030 GOTO 4000
[842] 4040 GOSUB 7350
[H2C] 4050 LOCATE 24, 14: PRINT "Wollen Sie diesen Date
nsatz löschen (J)a oder (N)ein ?";
[2C7] 4060 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 40
60
[V47] 4070 IF INSTR("jJnJ", TASTE$) = 0 THEN GOTO 4060
[6G6] 4080 IF INSTR("jJ", TASTE$) = 0 THEN RETURN
[UV4] 4090 FOR SCHLEIFE = 1 TO FELDER
[5L4] 4100 SATZ$(SATZ, SCHLEIFE) = ""
[8U2] 4110 NEXT SCHLEIFE
[V66] 4120 FOR SCHLEIFE = SATZ TO AKTUELL - 1
[GY4] 4130 FOR SCHLEIFE2 = 1 TO FELDER
[Ø29] 4140 SATZ$(SCHLEIFE, SCHLEIFE2) = SATZ$(SCHLE
IFE + 1, SCHLEIFE2)
[633] 4150 NEXT SCHLEIFE2
[WU2] 4160 NEXT SCHLEIFE
[6A7] 4170 IF AKTUELL <> 1 THEN AKTUELL = AKTUELL - 1
[1W1] 4180 RETURN
[4P0] 4190
[TK2] 4200 *****
[4S3] 4210 * Datei sortieren *
[1L2] 4220 *****
[FN0] 4230
[4H5] 4240 GOSUB 7230: GOSUB 7290: GOSUB 7350
[GW7] 4250 LOCATE 2, 26: PRINT "D A T E I S O R T I E
R E N"
[MV4] 4260 FOR SCHLEIFE = 1 TO FELDER
[3G6] 4270 LOCATE SCHLEIFE + 3, 2: PRINT SCHLEIFE
[1H8] 4280 LOCATE SCHLEIFE + 3, 4: PRINT "-"; INDEX$(
SCHLEIFE);
[BV2] 4290 NEXT SCHLEIFE
[Y1G] 4300 LOCATE 24, 3: PRINT "Nach welchem Schlüssel
soll ich die Datei sortieren ? Feld 1 bis"; FELDE
R; " ";
[268] 4310 Y = 24: X = 73: LAENGE = 2: MENGE$ = "zahl":
GOSUB 1880
[BT5] 4320 IF (TASTE$ = CHR$(27)) THEN RETURN
[3CB] 4330 IF (QS = "") OR (VAL(QS) = 0) OR (VAL(QS) >
FELDER) THEN GOSUB 1890: GOTO 4320
[RH4] 4340 LOCATE Y, X: PRINT QS; " ";
[RT2] 4350 SATZ = VAL(QS)
[DV3] 4360 GOSUB 7290: GOSUB 7350
[RU9] 4370 LOCATE 10, 23: PRINT "Die Daten werden nach
dem Kriterium"
[LF9] 4380 LOCATE 12, (80 - LEN(INDEX$(SATZ))) / 2: PRI
NT INDEX$(SATZ)
[GTA] 4390 LOCATE 14, 20: PRINT "sortiert. Bitte warten
Sie einen Moment."
[6W3] 4400 SZEIGER = AKTUELL - 1
[DM4] 4410 SZEIGER = INT(SZEIGER / 2)
[A84] 4420 IF SZEIGER = 0 THEN 4590
[Y92] 4430 AKTSATZ = 1
[9W5] 4440 HALBIERT = (AKTUELL - 1) - SZEIGER
[CU3] 4450 AKTZEIGER = AKTSATZ
[D25] 4460 SUMME = SZEIGER + AKTZEIGER
[UB9] 4470 IF SATZ$(AKTZEIGER, SATZ) < SATZ$(SUMME, SAT
Z) THEN 4560
[2W4] 4480 FOR SCHLEIFE = 1 TO FELDER
[H88] 4490 SATZ$(Ø, SCHLEIFE) = SATZ$(AKTZEIGER, SCHL
EIFE)
[Ø29] 4500 SATZ$(AKTZEIGER, SCHLEIFE) = SATZ$(SUMME,
SCHLEIFE)
[ØH7] 4510 SATZ$(SUMME, SCHLEIFE) = SATZ$(Ø, SCHLEIFE)
[5U2] 4520 NEXT SCHLEIFE
[2S5] 4530 AKTZEIGER = AKTZEIGER - SZEIGER
[FM4] 4540 IF AKTZEIGER < 1 THEN 4560
[EX1] 4550 GOTO 4460
[GX3] 4560 AKTSATZ = AKTSATZ + 1
[LT5] 4570 IF AKTSATZ <= HALBIERT THEN 4450
[GX1] 4580 GOTO 4410
[XYP] 4590 LOCATE 24, 5: PRINT "Der Sortiervorgang ist
beendet. Drücken Sie bitte eine beliebige Taste."
[914] 4600 WHILE INKEY$ = "": WEND
[MV1] 4610 RETURN
[RN0] 4620
```



```

[BA2] 4630 *****
[963] 4640 * Datenausgabe *
[LA2] 4650 *****
[7P0] 4660
[WH5] 4670 GOSUB 7230: GOSUB 7290: GOSUB 7350
[7L5] 4680 LOCATE 2, 34: PRINT "A U S G A B E"
[8YD] 4690 IF AKTUELL = 0 THEN Y = 10: FEHLER$ = "Datei
      noch nicht definiert": GOSUB 1760: RETURN
[143] 4700 Y = DATENAUSGABE
[0A6] 4710 LOCATE 7, 34: PRINT "Einzelne Sätze"
[XZ6] 4720 LOCATE 9, 32: PRINT "Ausgewählte Sätze"
[6X5] 4730 LOCATE 11, 36: PRINT "Alle Sätze"
[HN5] 4740 LOCATE 13, 36: PRINT "Hauptmenü"
[17G] 4750 LOCATE 24, 4: PRINT "Bitte wählen Sie mit de
      n Pfeiltasten aus und bestätigen Sie durch [Enter
      ]";
[0FD] 4760 COLOR 14, 1: FOR Q = 30 TO 50: LOCATE Y, Q:
      PRINT CHR$(SCREEN(Y, Q)): NEXT Q: COLOR 6, 0
[AH6] 4770 TASTE$ = INKEY$: IF TASTE$ = "" THEN 4770
[286] 4780 IF TASTE$ <> CHR$(0) + CHR$(72) THEN 4810
[BHD] 4790 IF Y <> 7 THEN FOR Q = 27 TO 53: LOCATE Y, Q
      : PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y - 2
[6X1] 4800 GOTO 4760
[A76] 4810 IF TASTE$ <> CHR$(0) + CHR$(80) THEN 4840
[YSD] 4820 IF Y <> 13 THEN FOR Q = 27 TO 53: LOCATE Y,
      Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y + 2
[KX1] 4830 GOTO 4760
[EE8] 4840 IF TASTE$ <> CHR$(13) THEN 4770 ELSE DATENAU
      SGABE = Y
[TX3] 4850 IF Y = 13 THEN RETURN
[XA3] 4860 Y2 = Y: GOSUB 7290
[043] 4870 Y = AUSDRUCKEN
[5S6] 4880 LOCATE 7, 33: PRINT "Nicht Ausdrucken"
[U86] 4890 LOCATE 9, 33: PRINT "Auch Ausdrucken"
[P65] 4900 LOCATE 11, 37: PRINT "Abbruch"
[R6G] 4910 LOCATE 24, 4: PRINT "Bitte wählen Sie mit de
      n Pfeiltasten aus und bestätigen Sie durch [Enter
      ]";
[PED] 4920 COLOR 14, 1: FOR Q = 30 TO 50: LOCATE Y, Q:
      PRINT CHR$(SCREEN(Y, Q)): NEXT Q: COLOR 6, 0
[UG6] 4930 TASTE$ = INKEY$: IF TASTE$ = "" THEN 4930
[586] 4940 IF TASTE$ <> CHR$(0) + CHR$(72) THEN 4970
[BGD] 4950 IF Y <> 7 THEN FOR Q = 30 TO 50: LOCATE Y, Q
      : PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y - 2
[VX1] 4960 GOTO 4920
[T76] 4970 IF TASTE$ <> CHR$(0) + CHR$(80) THEN 5000
[KSD] 4980 IF Y <> 11 THEN FOR Q = 30 TO 50: LOCATE Y,
      Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y + 2
[7Y1] 4990 GOTO 4920
[308] 5000 IF TASTE$ <> CHR$(13) THEN 4930 ELSE AUSDRUC
      KEN = Y
[1V5] 5010 IF Y = 11 THEN GOSUB 7290: GOTO 4700
[107] 5020 IF Y = 9 THEN DRUCKEN = 1 ELSE DRUCKEN = 0
[1V5] 5030 ON (Y2 - 1) / 2 - 2 GOSUB 5070, 5420, 6290
[VW1] 5040 GOTO 4640
[LN0] 5050
[7G2] 5060 *****
[RU3] 5070 * Einzelne Sätze *
[1H2] 5080 *****
[3P0] 5090
[LG5] 5100 GOSUB 7230: GOSUB 7290: GOSUB 7350
[2MB] 5110 LOCATE 2, 13: PRINT "A U S G A B E   E I N Z
      E L N E R   D A T E N S Ä T Z E"
[WWE] 5120 IF AKTUELL = 1 THEN Y = 10: FEHLER$ = "Es is
      t kein Datensatz definiert": GOSUB 1760: RETURN
[GAD] 5130 LOCATE 10, 16: PRINT "Welchen Satz wollen Si
      e ausgeben (maximal): AKTUELL - 1; " ?"
[978] 5140 Y = 10: X = 66: LAENGE = 5: MENGE$ = "zahl":
      GOSUB 1880
[LT5] 5150 IF (TASTE$ = CHR$(27)) THEN RETURN
[MVB] 5160 IF (Q$ = "") OR (VAL(Q$) = 0) OR (VAL(Q$) >
      AKTUELL - 1) THEN GOSUB 1890: GOTO 5150
[ZH4] 5170 LOCATE Y, X: PRINT Q$; " ";
[NT2] 5180 SATZ = VAL(Q$)
[NV3] 5190 GOSUB 7290: GOSUB 7350
[LXA] 5200 LOCATE 24, 49: PRINT "Satz"; SATZ; "von höch
      stens"; AKTUELL - 1;
[8C8] 5210 IF DRUCKEN = 1 THEN LPRINT "Satz"; SATZ: LPR
      INT
[W61] 5220 Y = 4
[478] 5230 LOCATE Y, 3: PRINT INDEX$(Y - 3); " "; SATZ$
      (SATZ, Y - 3)
[Z7A] 5240 IF DRUCKEN = 1 THEN LPRINT INDEX$(Y - 3); " ";
      SATZ$(SATZ, Y - 3)
[1G4] 5250 IF Y - 3 = FELDER THEN 5280
[TP1] 5260 Y = Y + 1
[BX1] 5270 GOTO 5230
[0U7] 5280 IF DRUCKEN = 1 THEN LPRINT : LPRINT : LPRINT
[RDB] 5290 LOCATE 24, 3: PRINT "(V)or oder (Z)urück ([E
      SC] für Abbruch) ? ---";
[AB7] 5300 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 53
      00
[HD8] 5310 IF INSTR("vVzZ" + CHR$(27), TASTE$) = 0 THEN
      GOTO 5300
[YC5] 5320 IF TASTE$ = CHR$(27) THEN RETURN
[LK6] 5330 IF INSTR("vV", TASTE$) = 0 THEN GOTO 5370

```

```

[816] 5340 IF SATZ = AKTUELL - 1 THEN GOTO 5300
[5U2] 5350 SATZ = SATZ + 1
[NX1] 5360 GOTO 5190
[5H4] 5370 IF SATZ = 1 THEN GOTO 5300
[RU2] 5380 SATZ = SATZ - 1
[1Y1] 5390 GOTO 5190
[BN0] 5400
[BW2] 5410 *****
[KR4] 5420 * Ausgewählte Sätze *
[LW2] 5430 *****
[VN0] 5440
[GH5] 5450 GOSUB 7230: GOSUB 7290: GOSUB 7350
[RG6] 5460 LOCATE 2, 10: PRINT "A U S G A B E   A U S G
      E W Ä H L T E R   D A T E N S Ä T Z E"
[TXE] 5470 IF AKTUELL = 1 THEN Y = 10: FEHLER$ = "Es is
      t kein Datensatz definiert": GOSUB 1760: RETURN
[4W4] 5480 FOR SCHLEIFE = 1 TO FELDER
[LG6] 5490 LOCATE SCHLEIFE + 3, 2: PRINT SCHLEIFE
[CG8] 5500 LOCATE SCHLEIFE + 3, 4: PRINT "-"; INDEX$(
      SCHLEIFE);
[PU2] 5510 NEXT SCHLEIFE
[8FG] 5520 LOCATE 24, 4: PRINT "Nach welchem Schlüssel
      soll ich die Datei durchsuchen ? Feld 1 bis"; FEL
      DER; " ";
[R78] 5530 Y = 24: X = 75: LAENGE = 2: MENGE$ = "zahl":
      GOSUB 1880
[VT5] 5540 IF (TASTE$ = CHR$(27)) THEN RETURN
[VCB] 5550 IF (Q$ = "") OR (VAL(Q$) = 0) OR (VAL(Q$) >
      FELDER) THEN GOSUB 1890: GOTO 5540
[7K4] 5560 LOCATE Y, X: PRINT Q$; " ";
[2X3] 5570 SCHLUESSEL = VAL(Q$)
[XV3] 5580 GOSUB 7290: GOSUB 7350
[42A] 5590 LOCATE 10, 21: PRINT "Geben Sie bitte das Su
      chkriterium ein"
[U40] 5600 Y = 12: X = 3: LAENGE = 75: MENGE$ = "alle":
      GOSUB 1880
[KT5] 5610 IF (TASTE$ = CHR$(27)) THEN RETURN
[R16] 5620 IF (Q$ = "") THEN GOSUB 1890: GOTO 5610
[MDC] 5630 IF (INSTR(Q$, ">") = 0) AND (INSTR(Q$, "<")
      = 0) AND (INSTR(Q$, "=") = 0) THEN Q$ = "=" + Q$
[2K4] 5640 LOCATE Y, X: PRINT Q$; " ";
[6BP] 5650 IF (INSTR(Q$, ">") > 1) OR (INSTR(Q$, "<") >
      1) OR (INSTR(Q$, "=") > 1) THEN Y = 20: FEHLER$
      = "< , > oder = müssen am Anfang stehen": GOSUB 17
      60: RETURN
[G34] 5660 ZEIGER = 0: ZAEHLER = 0
[4XD] 5670 ZEIGER = INSTR(ZEIGER + 1, Q$, "#"): IF ZEIG
      ER <> 0 THEN ZAEHLER = ZAEHLER + 1: GOTO 5670
[RZD] 5680 IF ZAEHLER > 2 THEN Y = 20: FEHLER$ = "Mehr
      als zwei '#' vorhanden": GOSUB 1760: RETURN
[5EN] 5690 IF ZAEHLER = 1 THEN IF (MID$(Q$, 2, 1) <> "#
      ") AND (RIGHT$(Q$, 1) <> "#") THEN Y = 20: FEHLER
      $ = "'#' an der falschen Position": GOSUB 1760: R
      ETURN
[8CP] 5700 IF ZAEHLER = 2 THEN IF (MID$(Q$, 2, 1) <> "#
      ") OR (RIGHT$(Q$, 1) <> "#") THEN Y = 20: FEHLER$
      = "'#' nicht an 2. und letzter Position": GOSUB
      1760: RETURN
[7V3] 5710 GOSUB 7290: GOSUB 7350
[N79] 5720 SATZ = 1: V$ = LEFT$(Q$, 1): IF ZAEHLER = 2
      THEN ZAEHLER = 3
[F39] 5730 IF ZAEHLER = 1 THEN IF RIGHT$(Q$, 1) = "#" T
      HEN ZAEHLER = 2
[Y38] 5740 Q$ = MID$(Q$, 2): IF LEFT$(Q$, 1) = "#" THEN
      Q$ = MID$(Q$, 2)
[SU7] 5750 IF RIGHT$(Q$, 1) = "#" THEN Q$ = LEFT$(Q$, L
      EN(Q$) - 1)
[145] 5760 IF SATZ <> AKTUELL THEN 5810
[ZV3] 5770 GOSUB 7290: GOSUB 7350
[YBE] 5780 LOCATE 24, 9: PRINT "Die Suche ist beendet.
      Drücken Sie bitte eine beliebige Taste.";
[579] 5790 WHILE INKEY$ = "" : WEND
[K24] 5800 RETURN
[SV1] 5810 IF V$ <> "<" THEN GOTO 5900
[Z94] 5820 IF ZAEHLER = 0 THEN IF Q$ <= SATZ$(SATZ, SCH
      LUESSEL) THEN GOTO 6090
[RBD] 5830 IF ZAEHLER = 2 THEN IF Q$ <= LEFT$(SATZ$(SAT
      Z, SCHLUESSEL), LEN(Q$)) THEN GOTO 6090
[YLD] 5840 IF ZAEHLER = 1 THEN IF Q$ <= RIGHT$(SATZ$(SA
      TZ, SCHLUESSEL), LEN(Q$)) THEN GOTO 6090
[X85] 5850 IF ZAEHLER <> 3 THEN GOTO 5890
[0H9] 5860 DA = 0: IF LEN(SATZ$(SATZ, SCHLUESSEL)) = 0
      THEN GOTO 5890
[T2V] 5870 FOR SCHLEIFE = 1 TO LEN(SATZ$(SATZ, SCHLUES
      SEL)): IF (Q$ <= MID$(SATZ$(SATZ, SCHLUESSEL), SCH
      LEIFE)) AND (LEN(Q$) <= LEN(MID$(SATZ$(SATZ, SCHL
      UESEL), SCHLEIFE))) THEN DA = 1
[WH6] 5880 NEXT SCHLEIFE: IF DA = 1 THEN GOTO 6090
[7Y1] 5890 GOTO 6030
[4B4] 5900 IF V$ <> ">" THEN GOTO 5990
[Z3B] 5910 IF ZAEHLER = 0 THEN IF Q$ >= SATZ$(SATZ, SCH
      LUESSEL) THEN GOTO 6090

```

Listing. »Parat« hält Ihre Daten zu jedem Zeitpunkt für Sie bereit


```
[SBD] 5920 IF ZAEHLER = 2 THEN IF QS >= LEFT$(SATZ$(SAT
Z, SCHLUESSEL), LEN(QS)) THEN GOTO 6090
[ZLD] 5930 IF ZAEHLER = 1 THEN IF QS >= RIGHT$(SATZ$(SA
TZ, SCHLUESSEL), LEN(QS)) THEN GOTO 6090
[V85] 5940 IF ZAEHLER < 3 THEN GOTO 5980
[YG9] 5950 DA = 0: IF LEN(SATZ$(SATZ, SCHLUESSEL)) = 0
THEN GOTO 5980
[03V] 5960 FOR SCHLEIFE = 1 TO LEN(SATZ$(SATZ, SCHLUESSEL)): IF (QS >= MID$(SATZ$(SATZ, SCHLUESSEL), SCH
LEIFE)) AND (LEN(QS) <= LEN(MID$(SATZ$(SATZ, SCHL
UESSEL), SCHLEIFE))) THEN DA = 1
[VH6] 5970 NEXT SCHLEIFE: IF DA = 1 THEN GOTO 6090
[6Y1] 5980 GOTO 6030
[Z4B] 5990 IF ZAEHLER = 0 THEN IF QS <> SATZ$(SATZ, SCH
LUESSEL) THEN GOTO 6090
[RAD] 6000 IF ZAEHLER = 2 THEN IF QS <> LEFT$(SATZ$(SAT
Z, SCHLUESSEL), LEN(QS)) THEN GOTO 6090
[YKD] 6010 IF ZAEHLER = 1 THEN IF QS <> RIGHT$(SATZ$(SA
TZ, SCHLUESSEL), LEN(QS)) THEN GOTO 6090
[MAC] 6020 IF ZAEHLER = 3 THEN IF INSTR(SATZ$(SATZ, SCH
LUESSEL), QS) = 0 THEN GOTO 6090
6030 GOTO 6110
[PW1] 6040 '
[HN0] 6050 '*****
[HH3] 6060 '* Datensatz nicht gefunden *
[385] 6070 '*****
[TH3] 6080 '
[LP0] 6090 SATZ = SATZ + 1
[EU2] 6100 GOTO 5760
[TW1] 6110 GOSUB 7290: GOSUB 7350
[RU3] 6120 LOCATE 24, 49: PRINT "Satz"; SATZ; "von höch
stens"; AKTUELL - 1;
[FC8] 6130 IF DRUCKEN = 1 THEN LPRINT "Satz"; SATZ: LPR
INT
[371] 6140 Y = 4
[BT8] 6150 LOCATE Y, 3: PRINT INDEX$(Y - 3); " "; SATZ$
(SATZ, Y - 3)
[68A] 6160 IF DRUCKEN = 1 THEN LPRINT INDEX$(Y - 3); " ";
SATZ$(SATZ, Y - 3)
[AF4] 6170 IF Y - 3 = FELDER THEN 6200
[0R1] 6180 Y = Y + 1
[TX1] 6190 GOTO 6150
[2T7] 6200 IF DRUCKEN = 1 THEN LPRINT : LPRINT : LPRINT
[62B] 6210 LOCATE 24, 3: PRINT "(W)eitersuchen ([ESC] f
ür Abbruch) ? -----";
[PB7] 6220 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 62
20
[CU7] 6230 IF INSTR("wW" + CHR$(27), TASTE$) = 0 THEN G
OTO 6220
[5D5] 6240 IF TASTE$ = CHR$(27) THEN RETURN
[P42] 6250 GOSUB 7350
[RX1] 6260 GOTO 6090
[3P0] 6270 '
[542] 6280 '*****
[033] 6290 '* Alle Sätze *
[832] 6300 '*****
[EN0] 6310 '
[3H5] 6320 GOSUB 7230: GOSUB 7290: GOSUB 7350
[GFA] 6330 LOCATE 2, 17: PRINT "A U S G A B E   A L L E
R   D A T E N S Ä T Z E"
[CXE] 6340 IF AKTUELL = 1 THEN Y = 10: FEHLER$ = "Es is
t kein Datensatz definiert": GOSUB 1760: RETURN
[4T1] 6350 SATZ = 1
[RX1] 6360 GOTO 5190
[6P0] 6370 '
[0M2] 6380 '*****
[9R3] 6390 '* Datei speichern *
[3L2] 6400 '*****
[HN0] 6410 '
[6H5] 6420 GOSUB 7230: GOSUB 7290: GOSUB 7350
[766] 6430 LOCATE 2, 32: PRINT "S P E I C H E R N"
[KXD] 6440 IF AKTUELL = 0 THEN Y = 10: FEHLER$ = "Datei
noch nicht definiert": GOSUB 1760: RETURN
[RG9] 6450 LOCATE 10, 3: PRINT "Geben Sie bitte den Dat
einamen ein:"
[8B8] 6460 Y = 10: X = 39: LAENGE = 40: MENGE$ = "alle"
: GOSUB 1880
[6U5] 6470 IF (TASTE$ = CHR$(27)) THEN RETURN
[DR5] 6480 IF QS = "" THEN GOSUB 1890: GOTO 6470
[KK4] 6490 LOCATE Y, X: PRINT QS; " ";
[2E8] 6500 LOCATE 24, 27: PRINT "Die Datei wird gespeic
hert";
[TG4] 6510 OPEN QS FOR OUTPUT AS #1
[RB3] 6520 PRINT #1, GRENZE
[C93] 6530 PRINT #1, FELDER
[3P4] 6540 FOR SCHLEIFE = 1 TO FELDER
[DZ4] 6550 PRINT #1, INDEX$(SCHLEIFE)
[K15] 6560 PRINT #1, LAENGE(SCHLEIFE)
[1K4] 6570 PRINT #1, ART$(SCHLEIFE)
[813] 6580 NEXT SCHLEIFE
[BD3] 6590 PRINT #1, AKTUELL
[HZ4] 6600 FOR SCHLEIFE = 1 TO AKTUELL
[9Z4] 6610 FOR SCHLEIFE2 = 1 TO FELDER
[RM6] 6620 PRINT #1, SATZ$(SCHLEIFE, SCHLEIFE2)
[BB3] 6630 NEXT SCHLEIFE2
```

```
[V03] 6640 NEXT SCHLEIFE
[MS1] 6650 CLOSE #1
[M3F] 6660 LOCATE 24, 9: PRINT "Die Datei ist gespeiche
rt. Drücken Sie bitte eine beliebige Taste.";
[924] 6670 WHILE INKEY$ = "": WEND
[MW1] 6680 RETURN
[RP0] 6690 '
[862] 6700 '*****
[8Y2] 6710 '* Datei laden *
[G62] 6720 '*****
[2P0] 6730 '
[RH5] 6740 GOSUB 7230: GOSUB 7290: GOSUB 7350
[625] 6750 LOCATE 2, 36: PRINT "L A D E N"
[4H9] 6760 LOCATE 10, 3: PRINT "Geben Sie bitte den Dat
einamen ein:"
[NB8] 6770 Y = 10: X = 39: LAENGE = 40: MENGE$ = "alle"
: GOSUB 1880
[LU5] 6780 IF (TASTE$ = CHR$(27)) THEN RETURN
[ZR5] 6790 IF QS = "" THEN GOSUB 1890: GOTO 6780
[UH4] 6800 LOCATE Y, X: PRINT QS; " ";
[000] 6810 OPEN "$." FOR OUTPUT AS #2: WRITE #2, HAUPT
MENUE, BEARBEITUNG, DATENAUSGABE, AUSDRUCKEN: CLO
SE : CLEAR : OPEN "$." FOR INPUT AS #2: INPUT #2
EN: CLOSE
[PD1] 6820 P = 39
[RS5] 6830 IF SCREEN(10, P) = 32 THEN GOTO 6860
[ZR5] 6840 QS = QS + CHR$(SCREEN(10, P)): P = P + 1
[LY1] 6850 GOTO 6830
[PP7] 6860 LOCATE 24, 29: PRINT "Die Datei wird geladen
";
[354] 6870 OPEN QS FOR INPUT AS #1
[UD3] 6880 INPUT #1, GRENZE
[FB3] 6890 INPUT #1, FELDER
[S0C] 6900 DIM SATZ$(GRENZE, FELDER), INDEX$(FELDER),
LAENGE(FELDER), ART$(FELDER)
[3P4] 6910 FOR SCHLEIFE = 1 TO FELDER
[DU5] 6920 LINE INPUT #1, INDEX$(SCHLEIFE)
[215] 6930 INPUT #1, LAENGE(SCHLEIFE)
[0L5] 6940 LINE INPUT #1, ART$(SCHLEIFE)
[813] 6950 NEXT SCHLEIFE
[DE3] 6960 INPUT #1, AKTUELL
[P05] 6970 FOR SCHLEIFE = 1 TO AKTUELL
[E05] 6980 FOR SCHLEIFE2 = 1 TO FELDER
[HG7] 6990 LINE INPUT #1, SATZ$(SCHLEIFE, SCHLEIF
E2)
[FA3] 7000 NEXT SCHLEIFE2
[ZZ2] 7010 NEXT SCHLEIFE
[SR1] 7020 CLOSE #1
[L9E] 7030 LOCATE 24, 9: PRINT "Die Datei ist geladen.
Drücken Sie bitte eine beliebige Taste.";
[D14] 7040 WHILE INKEY$ = "": WEND
[XW1] 7050 GOTO 1480
[VN0] 7060 '
[FA2] 7070 '*****
[L83] 7080 '* Programmende *
[RA2] 7090 '*****
[6N0] 7100 '
[VG5] 7110 GOSUB 7230: GOSUB 7290: GOSUB 7350
[Z07] 7120 LOCATE 2, 29: PRINT "P R O G R A M M E N D E
";
[DMD] 7130 LOCATE 10, 10: PRINT "Wollen Sie das Program
m wirklich verlassen (J)a oder (N)ein ?"
[8VE] 7140 LOCATE 24, 10: PRINT "Drücken Sie bitte entw
eder die Taste »J« oder die Taste »N«";
[9C7] 7150 TASTE$ = INKEY$: IF TASTE$ = "" THEN GOTO 71
50
[N57] 7160 IF INSTR("jJnN", TASTE$) = 0 THEN GOTO 7150
[EK6] 7170 IF INSTR("nN", TASTE$) <> 0 THEN RETURN
[X41] 7180 CLS
[8S1] 7190 KEY ON
[Z31] 7200 END
[DN0] 7210 '
[B03] 7220 '*****
[5M4] 7230 '* Obere Zeile löschen *
[L03] 7240 '*****
[XN0] 7250 '
[Z67] 7260 LOCATE 2, 2: PRINT STRING$(78, 32): RETURN
[5P0] 7270 '
[FY2] 7280 '*****
[M94] 7290 '* Bildschirm löschen *
[KX2] 7300 '*****
[GN0] 7310 '
[0ZA] 7320 FOR Y = 4 TO 22: LOCATE Y, 2: PRINT STRING$(
78, 32); : NEXT Y: RETURN
[SN0] 7330 '
[M43] 7340 '*****
[HT4] 7350 '* Untere Zeile löschen *
[W43] 7360 '*****
[8P0] 7370 '
[FG7] 7380 LOCATE 24, 2: PRINT STRING$(78, 32); : RETUR
N
```

Listing. »Parat« hält Ihre Daten zu jedem Zeitpunkt für Sie bereit

DOS-EXTRA DATABOX

– Die Software zur DOS-EXTRA Nr. 6 –

auf 2 Disketten im Format 5 1/4" oder
1 Diskette im Format 3 1/2"

sagenhaft günstig, extra viel gute Software fürs Geld!

DATABOX:

- mehr als der übliche Softwareservice
- bringt alle Programme der DOS EXTRA auf Datenträger

Für alle PCs unter MS-DOS als 5 1/4" - oder 3 1/2" -Diskette

Den Aufbau der Databox-Diskette finden Sie in der Datei »read.me« erklärt. Geben Sie deshalb vor dem Start »type read.me« ein.
Die Basic-Programme arbeiten mit dem GW-Basic-Interpreter von Microsoft einwandfrei. Bei Interpretern und Compilern anderer Hersteller kann es zu Problemen bei der Programmausführung kommen, sofern diese nicht voll kompatibel sind.



Databoxinhalte der DOS EXTRA Nr. 6

- SBasic-Compiler im Quelltext
- Datenbankprogramm »Parat«
- Fensterorientierte Funktionen für GW-Basic und SBasic
- Funktionen zur Stringverwaltung für GW-Basic und SBasic
- Turbo-Basic-Bibliothek für 16 Farben auf der CGA-Karte
- Hilfsfunktion für Basic-Grafikbefehle
- Programm zur Analyse von Grafikkarten
- Preview-Funktion für Dateien
- Maskengenerator zum Erzeugen von Binärdateien
- dBase-Datenbanken in Basic weiterverarbeiten
- Beispiele für rekursive Programmierung
- Superhirn für Quick Basic 4.0
- SChek 2.0: Checksummer für geprüfte Eingaben in Basic
- sowie alle begleitenden Listings zu den Rubriken Basic intern und Tips & Tricks

2 Disketten 5 1/4"
1 Diskette 3 1/2"

(Bestell-Nr. 3815)
(Bestell-Nr. 3816)

DM 35, – *

Noch erhältlich:

Databox DOS EXTRA Nr. 1

- SAM, der DOS-Assembler
- Clock, Uhr-Utilities
- Pascal – Include-Datei zum Aufruf von BIOS-Funktionen unter Turbo Pascal
- Tips und Tricks für jeden Zweck

Diskette 5 1/4" (Bestell-Nr. 350)
Diskette 3 1/2" (Bestell-Nr. 351)

Databox DOS EXTRA Nr. 2

- DOS-Commander, eine Benutzeroberfläche für MS-DOS
- SCopy, ein professionelles Kopierprogramm
- SForth 3.1, ein Compiler im Selbstbau
- SCAD, 3D-Grafik selbstgemacht
- Hurrican, ein Actionspiel der Superlative
- Supermind, ein Strategiespiel
- Utilities, Routinen für die tägliche Arbeit
- SChek – Checksummer für die Eingabe

4 Disketten 5 1/4" (Bestell-Nr. 352)
2 Disketten 3 1/2" (Bestell-Nr. 353)

Databox DOS EXTRA Nr. 3

- Datenfernübertragungsprogramm in 100% Assemblercode
- Herkules-Grafik-Toolbox in Turbo Pascal
- Grafikfunktionen für Turbo C
- Zeichensatzeditor für Epson-kompatible Drucker
- Treiber für AT-Tastatur
- alle begleitenden Listings zum Kurs »PC-glasklar«
- Demo für Sprachdigitalisierung
- ...und alle weiteren Listings

2 Disketten 5 1/4" (Bestell-Nr. 357)
1 Diskette 3 1/2" (Bestell-Nr. 358)

Databox DOS EXTRA Nr. 4

- Desktop-Publishing-Programm »SDTP«
- Kommandoeditor für MS-DOS
- Editor für Disketten und Festplatten
- Speicherresidente Programme auflisten und entfernen
- Paßwort-Schutz für MS-DOS
- Tastatureingabe automatisiert
- Clock-Utility für alle Grafikkarten
- Cursor und Zeichensatz für EGA-Karte programmieren
- Fensterfunktion für Turbo C
- Generator für Hilfsbildschirme
- Geprüfte Eingabe mit »SChek«

2 Disketten 5 1/4" (Bestell-Nr. 3811)
1 Diskette 3 1/2" (Bestell-Nr. 3812)

Databox DOS EXTRA Nr. 5

- 16 Farben für die CGA-Farbgrafikkarte
- Snapshot: Bildschirmfotos per Software
- Druckerzeichensatz selbst gemacht
- mausgesteuertes Zeichenprogramm
- Editor in GW-Basic
- Setup-Programm für ATs
- Maschinensprache für GW-Basic
- Erklärtes Listing: das Basic-Spiel »Stardos«
- Neuer Checksummer für geprüfte Eingabe

2 Disketten 5 1/4" (Bestell-Nr. 3813)
4 Disketten 3 1/2" (Bestell-Nr. 3814)

DM 24, – *

DM 45, – *

DM 35, – *

DM 35, – *

DM 35, – *

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 3, – DM bzw. für das Ausland 5, – DM Porto und Verpackung. – Unverbindliche Preisempfehlung – Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

Maskengenerator – Alternative zum Print-Befehl

Das Programm »mg_2« generiert aus einer mit einem Texteditor erzeugten Bildschirmmaske eine Binärdatei. Mit dem Basic-Befehl »bload« läßt sich diese Binärdatei in eigenen Programmen dann in Sekundenschnelle auf den Bildschirm bringen und stellt somit eine Alternative zum relativ langsamen Print-Befehl dar.

Stellen Sie sich vor, Sie wollen eine Bildschirmseite mit einer Eingabemaske oder einem Titelbild belegen, wozu normalerweise 25 Print-Anweisungen nötig wären. Abgesehen davon, daß »print« nicht unbedingt zu den schnellsten Basic-Befehlen gehört, benötigt ein derartiger Bildschirmaufbau viel Platz im Programm.

Das folgende Utility »mg_2.bas« (Listing) erleichtert das Anlegen von Eingabemasken, Titelbildschirmen und ähnlichem. Sie benötigen dazu einen beliebigen Texteditor, der ASCII-Dateien ohne Formatierungsanweisungen unterstützt. Die gewünschte Bildschirmmaske (im Programm als »map« bezeichnet) wird nun als normaler Text erstellt und mit der Namenserverweiterung »map« (zum Beispiel »titel.map«) gespeichert. Dabei sind maximal 23 Zeilen zu je 79 Zeichen zulässig.

Innerhalb der Bildschirmmaske sind Hervorhebungen durch Blinken, Anleuchten, Invertieren und Kombinationen daraus möglich. Sie markieren dazu mit Hilfe von Sonderzeichen den jeweiligen Bereich, der Maskengenerator setzt die Codes dann in entsprechende Bildschirmattribute um:

Blinken ein [Alt 240], aus [Alt 241]
Anleuchten ein [Alt 242], aus [Alt 243]
Invertieren ein [Alt 244], aus [Alt 245]
Blinken und Anleuchten ein [Alt 246], aus [Alt 247]
Blinken und Invertieren ein [Alt 248], aus [Alt 249]

Der gewünschte Text wird mittels »bload« blitzschnell in den Bildschirmspeicher Ihres PC geladen

Auf dem Bildschirm steht nach diesen Tastenkombinationen jeweils ein Sonderzeichen, das das Programm durch ein Null-Zeichen und das entsprechende Bildschirmattribut ersetzt. Verwenden Sie bitte keine Text-Formatierungsanweisungen Ihres Editors, also auch keine Tabulator-Zeichen.

Hat Ihre Bildschirmmaske den gewünschten Aufbau, dann starten Sie den Maskengenerator durch die Eingabe von
gwbasic mg_2

Dadurch lädt Ihr PC zuerst GW-Basic, dann das Programm »mg_2.bas« und startet es. Sie werden zunächst nach dem Dateinamen der Bildschirmmaske gefragt und geben diesen ohne die Erweiterung ».map« ein (Pfad-Angaben sind zulässig).

Im folgenden wird auf dem Bildschirm nach und nach die Maske aufgebaut und Zeilen, die mit Attribut-Steuerzeichen markiert wurden, blinken oder leuchten. Zum Schluß speichert das Programm den Bildschirminhalt mit der Namenserverweiterung ».bin« auf Diskette oder Festplatte. Von Ihren eigenen Basic-Programmen aus können Sie jederzeit eine so generierte Bildschirmmaske mit zwei Befehlen aufrufen:

```
def seg=&hb000 : bload "<Name>.bin",0
```

Dies bezieht sich allerdings auf eine Hercules-Grafikkarte. Bei einer Farbgrafikkarte müssen Sie »&hb800« anstelle von »&hb000« einsetzen. Ebenso ist in diesem Fall in der Zeile 800 des Listings die Segment-Adresse »&hb000« durch »&hb800« zu ersetzen.

Um einen wirklich blitzschnellen Bildschirmaufbau zu erreichen, sollten Sie die generierten Binärdateien auf einer RAM-Disk oder einer Festplatte speichern und auch in Ihrem eigenen Basic-Programm von dort laden (zum Beispiel »bload "c:\tmp\bild1.bin",0«).

(Detlef Drews/tr)

Name	mg_2.bas
Sprache	GW-Basic
Grafikkarte	jede
Besonderheiten	keine

```
[KD3] 10 'Programm: MG_2.BAS
[XW3] 20 'Maskengenerator
[8B3] 30 'Sprache: GW-Basic
[TR3] 40 'Autor: Detlef Drews
[U05] 50 'Copyright Everts & Hagedorn
[3D0] 60 '
[SD4] 70 'Maske senden und Eingabe
[ZV4] 80 'Bei nicht relevanter Maske
[255] 90 'Sprung zur Fehlerbehandlung
[3U1] 100 KEY OFF
[B41] 110 CLS
[FU1] 120 GOSUB 930
[933] 130 ON ERROR GOTO 860
[EN2] 140 MAP$="" : BIN$=""
[PT4] 150 LOCATE 7,33 : LINE INPUT MAP$
[KH0] 160 '
[BL5] 170 'Drücken von [ENTER] bedeutet
[3S2] 180 'Programmende
[LE5] 190 'Sonst Belegung der Variablen
[KR7] 200 IF LEN(MAP$)<1 THEN PRINT : PRINT : E
ND
[ZV2] 210 BIN$=MAP$+".BIN"
[964] 230 LOCATE 9,33 : PRINT BIN$
[MW2] 240 MAP$=MAP$+".MAP"
```

```
[HH0] 250 '
[V84] 260 'Öffnen der Map-Datei
[2B3] 270 'Speichersegment
[174] 280 'für Monochrom-Karte
[MA3] 290 'Variablen setzen
[8T2] 300 OPEN "I",#2,MAP$
[2HA] 310 DEF SEG=&HB000 : CLS : ZEILENZ=0 : POKEZAEHLE
R=-2 : MERKERBLINK=0
[YX2] 320 MERKERLEUCHT=0
[DZ2] 330 MERKERINVERS=0
[TW2] 340 MERKERBLEUCH=0
[QY2] 350 MERKERBINVER=0
[S75] 360 IF EOF(2) THEN CLOSE #2 : GOTO 710
[TH0] 370 '
[XZ3] 380 'Ein Zeichen einlesen
[Q25] 390 'Je nach Zeichen werden die
[G26] 400 'Merker gesetzt und das Zeichen
[284] 410 'bekommt den Wert CHR$(0)
[AX4] 420 POKEZAEHLER=POKEZAEHLER+2
[YM2] 430 XS=INPUT$(1,#2)
[DCB] 440 IF ASC(XS)=13 THEN XS=CHR$(0) : GOSUB 1090 :
POKE(POKEZAEHLER+3),0 : GOTO 360
[KC7] 450 IF XS=CHR$(240) THEN MERKERBLINK=1 : XS=CHR$(
0)
```



```
[VK7] 460 IF X$=CHR$(242) THEN MERKERLEUCHT=1 : X$=CHR$(0)
[6M7] 470 IF X$=CHR$(244) THEN MERKERINVERS=1 : X$=CHR$(0)
[CL7] 480 IF X$=CHR$(246) THEN MERKERBLEUCH=1 : X$=CHR$(0)
[4N7] 490 IF X$=CHR$(248) THEN MERKERBINVER=1 : X$=CHR$(0)
[5C7] 500 IF X$=CHR$(241) THEN MERKERBLINK=0 : X$=CHR$(0)

[DK7] 510 IF X$=CHR$(243) THEN MERKERLEUCHT=0 : X$=CHR$(0)
[U9B] 520 IF X$=CHR$(245) THEN MERKERINVERS=0 : X$=CHR$(0) : POKE POKEZAEHLER+1,0
[VK7] 530 IF X$=CHR$(247) THEN MERKERBLEUCH=0 : X$=CHR$(0)
[SAB] 540 IF X$=CHR$(249) THEN MERKERBINVER=0 : X$=CHR$(0) : POKE POKEZAEHLER+1,0

[RH0] 550 '
[M25] 560 'Poken des ASCII-Wertes des
[YS3] 570 'gelesenen Zeichens
[M75] 580 'Poken des Attributes je nach
[RE3] 590 'Stand des Merkers
[7G4] 600 POKE POKEZAEHLER,ASC(X$)
[WE7] 610 IF MERKERBLINK=1 THEN POKE POKEZAEHLER+3,135
[ZN7] 620 IF MERKERLEUCHT=1 THEN POKE POKEZAEHLER+3,15
[WU7] 630 IF MERKERINVERS=1 THEN POKE POKEZAEHLER+3,112
[XR7] 640 IF MERKERBLEUCH=1 THEN POKE POKEZAEHLER+3,143
[BT7] 650 IF MERKERBINVER=1 THEN POKE POKEZAEHLER+3,240
[9R1] 660 GOTO 360
[2H0] 670 '
[XA5] 680 'Vorsichtshalber die rechten
[BZ4] 690 'Randattribute auf den Wert
[YU1] 700 '0 setzen
[XD2] 710 POKEWERT=-1
[2T2] 720 FOR LX=1 TO 24
[734] 730 POKEWERT=POKEWERT+160
[043] 740 POKE POKEWERT,0
[PS2] 750 NEXT LX : LX=0
[YH0] 760 '
[P75] 770 'Speichern der Bildschirm-Map
[UR5] 780 '(binär) von Zeile 1 bis Zeile 23
[KY6] 790 'und Rücksprung zum Programmanfang
[9M2] 800 DEF SEG=&HB000
[283] 810 BSAVE BIN$,0,&HE5D0
[3P1] 820 GOTO 100
[RH0] 830 '
[FE3] 840 'Fehlermeldung bei
[E34] 850 'nicht vorhandener Map
[PC2] 860 PRINT CHR$(7)
[V0C] 870 LOCATE 20,1 : PRINT " ***** Unter diesem N
amen existiert keine Map *****"

[2N4] 880 FOR WX=1 TO 1000 : NEXT WX
[WM1] 890 CLOSE #2
[002] 900 RESUME 100
[LH0] 910 '
[NW2] 920 'Eingabe - Maske
[151] 930 CLS
[8Y8] 940 PRINT "

[D4A] 950 PRINT " | BASIC - M A S K E
N - G E N E R A T O R | "
[WF8] 960 PRINT " | "

[ASC] 970 PRINT " | Autor: Detlef Drews
Copyright Everts & Hagedorn | "
[2YA] 980 PRINT " |-----| "

[5G8] 990 PRINT " | "

[20B] 1000 PRINT " | Masken-Dateiname [.map] :
| "
[2V8] 1010 PRINT " | "

[0SA] 1020 PRINT " | Bild-Dateiname [.bin] :
| "
[SG9] 1030 PRINT " |-----| "

[8V1] 1040 RETURN
[BN0] 1050 '
[CW5] 1060 'Weiterzählen des Pokezahlers
[985] 1070 'auf volle Zeilen (80 Zeichen)
[KS4] 1080 'bzw. mit Attribut auf 160
[6C9] 1090 IF INT(POKEZAEHLER/160)=(POKEZAEHLER/160) TH
EN RETURN
[881] 1100 LX=1
[1D5] 1110 POKEZAEHLER=POKEZAEHLER+LX
[75E] 1120 IF INT(POKEZAEHLER/160)=(POKEZAEHLER/160) TH
EN POKEZAEHLER=POKEZAEHLER-4 : RETURN
[TV1] 1130 GOTO 1100
```

Listing. »mg_2.bas«, ein Maskengenerator für GW-Basic



TOOLBOX-Spezial

Spezial I: Turbo Prolog

Eine Toolbox mit Routinen zur "Mustererkennung", je einem Programm für statistische Auswertungen und analytisches Integrieren und Differenzieren mathematischer Themen, DBase-Dateien als Prolog-Wissensbasis und ein Turbo Prolog-Tracer. Dokumentationen zu den Programmen auf der Diskette.

IBM 5 1/4 Zoll, Best.- Nr. 2801
3 1/2 Zoll, Best.- Nr. 2802

35,- DM*

Spezial II: PasComp - Standard Pascal Compiler

Komplettes Echtzeitsystem zu Studienzwecken in Turbo Pascal 3.0 Quellcode: Scheduler, Debugger, Utilities und Beispiele. Anpassungshinweise an andere Compiler, Programmiertechniken und Grundlagen der Echtzeitverarbeitung als Dokumentation auf der Diskette. Auch mit Turbo Pascal 4/5 kompilierbar.

IBM 5 1/4 Zoll, Best.- Nr. 2804
3 1/2 Zoll, Best.- Nr. 2805
Atari ST 3 1/2 Zoll, Best.- Nr. 2803

35,- DM*

Spezial III: Echtzeitverarbeitung

Komplettes Echtzeitsystem zu Studienzwecken in Turbo Pascal 3.0 Quellcode: Scheduler, Debugger, Utilities und Beispiele. Anpassungshinweise an andere Compiler, Programmiertechniken und Grundlagen der Echtzeitverarbeitung als Dokumentation auf der Diskette. Auch mit Turbo Pascal 4/5 kompilierbar.

IBM 5 1/4 Zoll, Best.- Nr. 2806
3 1/2 Zoll, Best.- Nr. 2807

35,- DM*

Spezial IV: CALC & Kurvendiskussion

Mit dem Mathe-Compiler CALC können Formeln und Funktionen zur Auswertung direkt über die Tastatur eingegeben werden. Jetzt auch Differenzieren! Lauffähiges Demonstrationsprogramm "Kurvendiskussion" mit Bildschirmmodulen für Apple II, IBM-CGA und EGA. Komplette Quellcodes in Turbo Pascal 3.0. Mit Dokumentation auf Diskette.

IBM 5 1/4 Zoll, Best.- Nr. 2808
3 1/2 Zoll, Best.- Nr. 2809

35,- DM*

Spezial V: STRUKTO Plus "Pascal"

Erzeugung von Strukturdateien, grafischen Struktogrammen und Programmorganisationsplänen (POP) aus Pascal-Quelltexten oder Strukturdateien. Ausgabe wahlweise auf Drucker, Bildschirm oder in Dateien, inklusive einer Fehleranalyse. Turbo Pascal 3.0 Quellcode, problemlos auch für Turbo Pascal 4/5 verwendbar. Anleitung und Dokumentation auf der Diskette.

IBM 5 1/4 Zoll, Best.- Nr. 2811
3 1/2 Zoll, Best.- Nr. 2812

35,- DM*

Spezial VI: Spiel & Spaß

Die besten Spiele und Knobels aus PASCAL und TOOLBOX jetzt in Farbe: Stratego, Schiebung, Inverso, der Zahlenwurm... Dazu ein kompletter Sprite-Editor für eigene Ausgestaltungen und eine Turbo Pascal "Spiele-Unit" mit vielen grundlegenden Funktionen. Alles als Turbo Pascal 4.0 Quellcode und als lauffähige Programme. Anleitungen als Datenfile auf Diskette.

IBM 5 1/4 Zoll, Best.- Nr. 2813
3 1/2 Zoll, Best.- Nr. 2814

35,- DM*

Spezial VII: Deutsche Silbentrennung

Tabellengesteuerte deutsche Silbentrennung mit mehr als 99% Treffsicherheit: Dokumentierte Algorithmen und Turbo Pascal 3.0 (auch 4/5) Quellcode. Ein lauffähiges Programm fügt beliebige Trennzeichen in einen ASCII-Text ein. Dazu passend ein System zum Aufbau beliebiger großer Wörterbücher. Quellcode in Modula-2 und demonstrierte Anwendungen. Dokumentation auf Diskette.

IBM 5 1/4 Zoll, Best.- Nr. 2815
3 1/2 Zoll, Best.- Nr. 2816

35,- DM*

Die Spezial's III, IV, V und VII liefern wir auf Wunsch auch für ausgesuchte CPM/Format und Apple II mit CPM/Karte! Genaue Informationen bietet das Merkblatt "CPM-Databox", das Sie mit adressiertem Rückumschlag beim Softwarevertrieb des DMV-Verlages anfordern können.

* Unabhängig von der Anzahl der bestellten Artikel berechnen wir für das Inland 3,- DM bzw. für das Ausland 5,- DM Porto und Verpackung. Unverbindliche Preisempfehlung

Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

Die Speicherverwaltung von GW-Basic

Der GW-Basic-Interpreter Ihres PC verwendet den vorgesehenen Arbeitsspeicher nach einem bestimmten Schema. Dabei reserviert er zum Beispiel bestimmte Bereiche im BIOS-Datenbereich für seine Zwecke und richtet sich einen Arbeitsbereich im Hauptspeicher des Computers ein.

Prinzipiell belegt der GW-Basic-Interpreter Speicherplatz in zwei unterschiedlichen Bereichen des Arbeitsspeichers. Für das eigentliche Basic-Programm und seine eigenen Routinen nutzt er den Arbeitsspeicher, den MS-DOS für die Verwendung durch Anwenderprogramme bereitstellt. Zusätzlich dienen die Interruptvektoren 80hex bis F0hex als Zwischenspeicher für diverse Informationen und Zeiger auf bestimmte Funktionen.

Im Bereich von Adresse 510hex bis 51Dhex, der auf das BIOS-Datensegment folgt, legt der GW-Basic-Interpreter einige Informationen ab, die für die korrekte Ausführung eines Programms notwendig sind. Zum Beispiel »merkt« es sich in den Speicherzellen 510hex und 511hex die Adresse des Basic-Datensegments, in dem sich das Basic-Programm samt Variablen befindet.

Im Arbeitsspeicherbereich von MS-DOS, dessen Startadresse von der Größe der installierten residenten Programme und von der DOS-Version abhängt, steht zunächst der Programmcode des Basic-Interpreters, gefolgt von einem weiteren Datenbereich, in dem unter anderem der Puffer für die serielle RS232-Schnittstelle enthalten ist.

Neben den Send- und Empfangsdaten werden auch die Routinen für die Ansteuerung der RS232-Schnittstelle in diesem Bereich abgelegt. Das hat folgenden einfachen Grund: Der GW-Basic-Interpreter läßt sich mit der Option »/c:0« aufrufen. Die Routinen

für die Ansteuerung der RS-232-Schnittstelle und der Pufferspeicher werden dann nicht installiert und belegen daher keinen wertvollen Speicherplatz.

Auf den Schnittstellenpuffer folgt das Listing des Basic-Programms, jedoch nicht als ASCII-Text, sondern in der sogenannten Token-Schreibweise. Dabei werden die Zeilennummern und numerischen Werte in einem bestimmten Format gespeichert.

Auch für jedes Basic-Schlüsselwort, gleichgültig ob es sich um eine Basic-Funktion oder um einen Basic-Befehl handelt, existiert eine Sequenz von ein oder zwei Byte Länge. Dadurch spart der GW-Basic-Interpreter einerseits Speicherplatz und beschleunigt andererseits die Programmausführung.

Die Umwandlung der eingegebenen ASCII-Zeichen in die Token-Schreibweise erledigt der Interpreter immer dann, wenn Sie ihre Eingaben oder Änderungen mit der Entertaste abschließen.

Oberhalb des mit Programmcode belegten Speicherbereichs sind alle einfachen Variablen außer den Zeichenketten gespeichert. Danach folgen die Feldvariablen (Arrays).

Am oberen Ende des Basic-Datensegments liegt der Basic-Stapel, auf dem Rücksprungradressen für Gosub-Befehle gesichert und Parameter übergeben werden. Unterhalb des Basic-Stapels sind die Zeichenketten-Variablen (Strings) abgelegt.

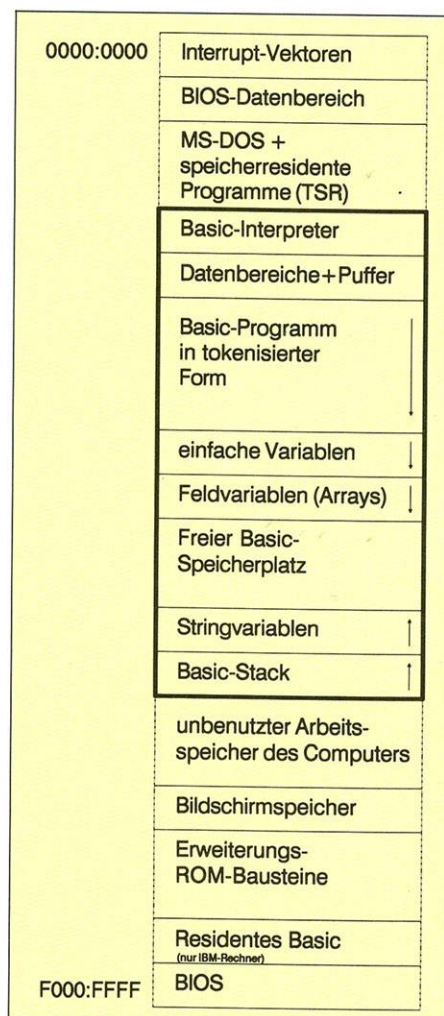
Die Größe all dieser Speicherbereiche ist nicht festgelegt, weil Länge und

Anzahl von Variablen nicht vor dem Programmstart bekannt sind.

Im Falle des Basic-Stapels und der RS232-Routinen kann der Anwender sie beim Aufruf von GW-Basic selbst festlegen und die Speicheraufteilung dadurch seinen Bedürfnissen anpassen.

Die anderen Bereiche ändern ihre Größe dynamisch. Dabei bewegen sich einfache Variablen und Feldvariablen vom unteren Ende des Basic-Datensegments hinauf in Richtung oberes Ende, und der Basic-Stapel und die Zeichenketten-Variablen dehnen sich vom oberen Ende des Basic-Datensegments kommend nach unten aus. Daher ergibt sich der freie Speicherplatz, den die Basic-Funktion »fre(0)« oder »fre(“ “) zurückliefert, aus dem freien Bereich zwischen Feldvariablen und Stringvariablen. Das Bild zeigt den Aufbau des Arbeitsspeichers eines PC, wenn der GW-Basic-Interpreter geladen ist.

(Martin Hepp/sw)



So ist der Arbeitsspeicher des PC belegt, wenn der GW-Basic-Interpreter geladen ist

Die GW-Basic-Variablen im Arbeitsspeicher

Adresse	Bedeutung
0050:0010 bis 0050:0011	Segmentadresse des Basic-Datensegments
0050:0012 bis 0050:0013	Segmentadresse der Systemuhr-Routine
0050:0014 bis 0050:0015	Offsetadresse der Systemuhr-Routine
0050:0016 bis 0050:0017	Segmentadresse der [Ctrl-Break]-Routine
0050:0018 bis 0050:0019	Offsetadresse der [Ctrl-Break]-Routine
0050:001A bis 0050:001B	Segmentadresse der Diskettenfehler-Routine
0050:001C bis 0050:001D	Offsetadresse der Diskettenfehler-Routine

Der GW-Basic-Interpreter speichert verschiedene Informationen im Adreßbereich von 510hex bis 51Dhex ab

PC-SPIELEBOX NO.1

BIMBO II

Das Geschicklichkeitsspiel für den PC überhaupt!

Wie in einem Squashcourt spielen Sie einen Ball von rechts nach links durch einen auf drei Seiten geschlossenen Bildschirm. Auf der offenen Seite müssen Sie mit Ihrem Schläger verhindern, daß der Ball ins Aus geht. Im Spielfeld selbst befinden sich unterschiedliche Hindernisse, die jedes für sich, unterschiedliche Auswirkungen auf den weiteren Spielverlauf haben. So gibt es Blöcke, die dem Spieler, sobald sie getroffen wurden, einen Extraball gutschreiben. Andere wiederum ziehen einen Ball ab, zählen Bonus, verschnellern das Spiel und und und...

Ist es Ihnen gelungen, alle Hindernisse zu beseitigen, gelangen Sie in das nächste Bild von Bimbo, das Sie wiederum mit neuen Hinderniskonstellationen konfrontiert.

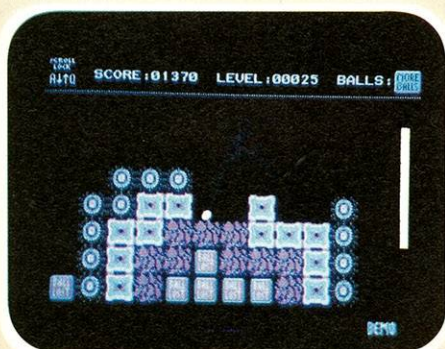
Aber Bimbo ist mehr als nur ein Spiel. Mittels des zum Programm gehörenden Editors sind Sie in der Lage, sich genau die Bilder zusammenzusetzen, die Sie gerne spielen möchten.

BOUNCER

Bouncer versetzt Sie hinter das Lenkrad eines ganz besonderen Automobils. Das Bounce'O'mobil kann springen. Daß es dies nicht nur aus Spaß an der Freude tut, wird spätestens beim ersten Felsbrocken klar, der Ihnen den Weg versperrt. Nun sind Sie gefordert, überspringen Sie das Hindernis. Jedoch werden Sie nicht nur durch herumliegende Findlinge gehandicapt. So manche Brücke, die Sie auf Ihrer Fahrt überqueren müssen, ist eingestürzt, tollwütige Vögel versuchen, sich aus der Luft auf Sie zu stürzen, Straßensperren blockieren den Weg, und der rachsüchtige Pilot einer Sportmaschine stellt Ihnen nach.

Wird es Ihnen gelingen, die Straße bis zu ihrem Ende zu fahren?

Bouncer, die PC-Variante eines erfolgreichen Arcade Automaten, wurde 100% in Maschinencode geschrieben.



Best.-Nr. 5 1/4" Disk. 129
Für MS-DOS erhältlich

Best.-Nr. 3 1/2" Disk. 185

49,-DM*

PC-SPIELEBOX NO.2

JACKY

Hier erwartet Sie ein Geschicklichkeitsspiel der besonderen Art.

Sie steuern einen kleinen Roboter, der von Ebene zu Ebene springen kann. Allerdings stehen diese unter Strom, so daß Jacky versuchen muß, durch die Lücken zu springen, ohne die Ebenen von unten zu berühren. Dabei gilt es natürlich, möglichst nicht die vielen Monster zu berühren, die Jacky das Leben schwer machen. Auf der anderen Seite können Sie auf Ihrem Weg nach oben eine ganze Reihe von Boni einsammeln, die Ihren Score gehörig aufpolieren können. Programmiert von Hartmut Pfarr, dem bekannten DMV-Autor, verspricht Jacky ein echter Renner zu werden. Versäumen Sie also auf keinen Fall die aufregenden Abenteuer des kleinen Roboters Jacky.

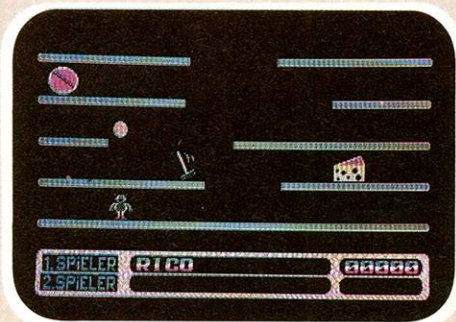
Die Abenteuer des kleinen Jacky sind im System MS-DOS erhältlich.

STARVISION

Ein exzellentes Shoot'em up auf Ihrem PC!

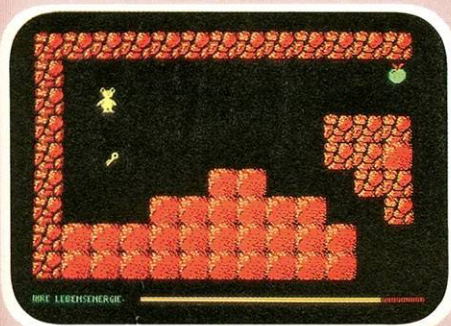
Besteigen Sie Ihr Schiff und retten Sie den Mond. Außerirdische Kreaturen haben sich dort eingenistet und behindern die Aktivitäten der Menschen. Deshalb ist es jetzt Ihre Aufgabe, den Mond von diesen Wesen zu befreien. Die größte Hilfe bei Ihrer Aufgabe ist Ihr kleines Raumschiff, daß Sie im Verlauf dieses Spieles mit zahlreichen Extrafeatures ausrüsten können. Trotzdem keine einfache Aufgabe, da die Außerirdischen bereits mehrere Stützpunkte eingerichtet haben. Wenn Sie den ersten Sektor gesäubert haben, bekommen Sie einen Bonus und können sich am nächsten versuchen. Starvision verspricht somit viele Stunden spannender Unterhaltung.

Starvision ist im System MS-DOS erhältlich.



Best.-Nr. 5 1/4" Disk. 1291 Best.-Nr. 3 1/2" Disk. 1292

49,-DM*



5 1/4" Disk. Best.-Nr. 186
3 1/2" Disk. Best.-Nr. 187

49,-DM*

Underground Strategie & Action für Ihren PC!

Jetzt neu

Wer denkt, daß unter der Erde tiefer Friede herrscht, der irrt sich gewaltig. Unbemerkt von uns, den oberirdischen Betrachtern, spielt sich unterhalb der Erdoberfläche so manches Drama im Verborgenen ab.

Underground, so der Titel des neuesten Hits von DMV, beleuchtet die Vorgänge unter der Erde. Begleiten Sie den kleinen Maulwurf, den Star dieses Programms, bei seinem heroischen Kampf gegen seine greulichen Feinde, und helfen Sie ihm, den Ausweg aus den finsternen und lichtlosen Katakomben des Schreckens zu finden.

Doch erwarten Sie keine langweiligen Grabespiele, wie man bei einem Maulwurf vermuten könnte! Spätestens, seit prominente Frösche Eigentumswohnungen bewohnen, haben sich die Standards im Tierleben durchgreifend geändert.

Maulwürfe, die up to date sind, haben Ihr Labyrinth beispielsweise mit hochmodernen Beam-Stationen ausgerüstet, die einen zeitsparenden Transport zwischen abgelegenen Teilen des Systems ermöglichen.

In Actioneinlagen müssen Feinde in ihre Schranken verwiesen werden, die den kleinen Maulwurf zum Fressen gern haben. Eine Landkarte erleichtert die Orientierung innerhalb der jeweiligen Labyrinth-Ebene, diese Karte muß allerdings erst gefunden werden. Bei der Suche ist so mancher Gegenstand zu finden, der das Abenteuer erleichtern kann, so gibt es Nahrung, die die schwindenden Kräfte wieder aufstockt, eine Fackel, mit der lichtscheues Gesindel vertrieben werden kann, kleine Items, die den Maulwurf für kurze Zeit zum unbesiegbaren Superhelden machen, und, und, und.

Wenn Sie Streit mit potentiellen Gegnern anfangen, können Sie Ihre Zielgenauigkeit im Steinwurf unter Beweis stellen, um Ihre Feinde Mores zu lehren.

Wird es Ihnen gelingen, den kleinen Maulwurf aus den Katakomben des Schreckens zu befreien?

Die Abenteuer des kleinen Maulwurfes sind für das System MS-DOS (CGA) erhältlich.

KNOW-PC

Das Spiel für die ganze Familie!

JETZT NEU!

Die Know-PC-Ergänzungsdiskette:
20 neue Wissensgebiete
5 1/4"-Disk Best.-Nr. 1293
3 1/2"-Disk Best.-Nr. 1294

29,-*



BLUTRACHE

SUEDAMERIKANISCHES,
KLEINES NAGETIER

WAS IST
Vendetta ?

MARTIN 3000
ABINE 3000
ERND 3500
OGI 3000

MARTIN, DU
BIST AN DER
REIHE !

KNOW-PC - Frage- und Antwortspiel der Extraklasse

- kann eine unbegrenzte Anzahl von Fragen und Antworten verwalten!
- bietet die Möglichkeit, eigene Fragen in einem selbstgewähltem Wissensgebiet einzugeben, und zwar mit einem komfortablen Editor!
- besitzt eine Supergrafik!
- ist spielbar mit 1 bis 4 Einzelspielern oder in Gruppen mit einem Vielfachen davon!
- muß man mit Strategie spielen, da der Beste in jedem Wissensgebiet am Ende noch Zusatzpunkte bekommt!
- verlangt keinerlei Programmierkenntnisse!
- wird mit 13 verschiedenen Wissensgebieten ausgeliefert!
- kann auch für andere Zwecke Verwendung finden, z.B. als Vokabeltrainer

Konfiguration:

PC XT/AT mit 512 KByte oder mehr, 1 Diskettenlaufwerk oder Festplatte, MS-DOS ab Version 2.0 oder größer, für Farbgrafik, Monochrom, Hercules oder EGA.

5 1/4"-Disk Best.-Nr. 161
3 1/2"-Disk Best.-Nr. 162

49,-DM*

* Unabhängig von der Anzahl der bestellten Programme berechnen für das Inland 3,- DM bzw. für das Ausland 5,- DM Porto und Verpackung. - Unverbindliche Preisempfehlung - Bitte benutzen Sie die Bestellkarte.

Bücher
Software
Zeitschriften
Postfach 250
3440 Eschwege

DMV
Daten- und
Medienverlag

Variablen speichern unter GW-Basic

Basic bietet dem Programmierer unterschiedliche Typen von Variablen an. Für eine optimale Ausnutzung der Fähigkeiten des Interpreters und die maschinennahe Programmierung unter Basic ist die Kenntnis der internen Speicherformate von Basic-Variablen und ihres Inhalts sehr nützlich.

In GW-Basic werden die verschiedenen Variablentypen durch besondere Symbole hinter dem Variablennamen gekennzeichnet. Anstelle der »klassischen« Unterteilung der Variablen in numerische und alphanumerische Werte (Zahlen und Zeichenketten) können Sie in GW-Basic für die numerischen Variablen das Format der Daten noch etwas genauer angeben. Neben den Zeichenketten stehen drei unterschiedliche Zahlentypen zur Verfügung: die Ganzzahl, die einfache genaue Fließkommazahl und die doppelte genaue Fließkommazahl. Der Zahlentyp, der einer Variablen zugewiesen werden darf, wird durch ein Typensymbol hinter dem Variablennamen gekennzeichnet. Tabelle 1 zeigt die vier unterschiedlichen Datentypen und ihr Typensymbol.

Die Variablen werden vom GW-Basic-Interpreter gemeinsam mit dem Programmtext in einem einzigen Speichersegment abgelegt. Unmittelbar auf interne Pufferbereiche des Interpreters (für Dateizugriff und Schnittstellenverwaltung) folgt der Programmtext in Token-Codierung. Hinter dem Programmtext beginnt der Variablenbereich für alle numerischen Variablen, also Ganzzahlen, einfache und doppelte Fließkommazahlen sowie ein- oder mehrdimensionale Felder dieser Variablentypen.

Der GW-Basic zur Verfügung stehende Speicherbereich ist nach einem festen Schema aufgeteilt

Je mehr numerische Variablen in einem Programm definiert werden, desto weiter wächst der von ihnen belegte Platz im Basic-Speichersegment von unten nach oben.

Am oberen Ende des Basic-Segments befindet sich der Stack (Stapel). Auf ihm werden Rücksprungrückadressen für Schleifen und Unterprogrammaufrufe sowie andere temporäre Daten gespeichert. Der Speicher unterhalb des Stacks wird zum Ablegen von Zeichenketten benutzt. Dieser Bereich wächst von oben nach unten, das heißt, mit zunehmender Größe des Variablenbereichs für Zeichenketten

werden die Speicheradressen immer kleiner.

Auf diese Weise kommen sich also der numerische Variablenspeicher und der String-Variablenspeicher entgegen. Sobald sich die Grenzen der beiden Speicherbereiche berühren, ist kein freier Speicher mehr verfügbar: Der Interpreter meldet einen Fehler. Damit dies unterbleibt, »räumt« der Interpreter von Zeit zu Zeit den String-Variablenbereich mit der sogenannten »Garbage Collection« auf, um die Löcher im Speicher, die durch gelöschte Zeichenketten entstehen, mit Zeichenketten, die sich im unteren Bereich des Speichers befinden, zu füllen.

Mit der »Garbage Collection« räumt der Interpreter den Variablenspeicher für Zeichenketten auf

Um das Verfahren der Garbage Collection und ihre Notwendigkeit zu verstehen, müssen Sie sich das Speichern von Variablen etwas genauer betrachten. Das Speichern numerischer Daten ist für den Interpreter relativ einfach. Jede numerische Variable belegt im Speicher, je nach Datentyp, eine feste Anzahl von Bytes. Die Bytezahl definiert den Wertebereich für eine Variable, in dem sich der Inhalt dieser Variablen bewegen darf. Eine Ganzzahl (Integer) belegt zum Beispiel immer 16 Bit, also zwei Byte, gleichgültig, welchen aktuellen Wert sie zugewiesen bekommt. Man spricht in diesem Zusammenhang auch von »statischen Variablen«. Ändert sich der Wert einer statischen Variablen, so sucht der Interpreter ihre Adresse aus seinen internen Zeigern und schreibt den neuen Variableninhalt in die für diese Variable vorgesehenen Speicheradressen. Wird eine neue Variable definiert, so muß für sie lediglich ein noch unbelegter Bereich am oberen Ende des Variablenspeichers reserviert werden.

Anders verhält es sich beim Speichern von Zeichenketten. Hier ist für die Speicherung der Daten ein erheblich größerer Verwaltungsaufwand

erforderlich, denn im Gegensatz zu den numerischen Variablen besitzen String-Variablen keine statische, sondern eine dynamische Länge. Nach der Zuweisung

```
n$="kurz"
```

belegt beispielsweise der Inhalt der Variablen »n\$« vier Byte, nach der Zuweisung

```
n$=n$+"lang"
```

jedoch acht Byte. Im Verlauf eines Programms können sich also die Längen der String-Variablen fast beliebig ändern. Eine Grenze ist lediglich durch die maximal mögliche Länge von 255 Zeichen für eine Zeichenkette gegeben.

Wie soll der Interpreter nun aber die dynamischen Variablen verwalten? Die einfachste Möglichkeit besteht darin, für jede neue String-Variable direkt 255 Byte Speicherplatz zu reservieren. Dann muß sich der Interpreter keine Sorgen um die Lage der Variablen machen: Wie bei statischen Variablen bleibt die Adresse einer String-Variablen immer konstant. Allerdings würde bei diesem Verfahren der Arbeitsspeicher des Computers kolossal verschwenden, weil auch sehr kurze Variablen immer 256 Byte Speicher belegen würden. Vor allem in Anbetracht der Tatsache, daß unter GW-Basic nur ein einziges Segment für die Speicherung von Programm und Daten zur Verfügung steht, wäre dieses Verfahren völlig indiskutabel.

Der Compiler »SBasic« aus dieser Ausgabe der DOS Extra verwendet dagegen das beschriebene Verfahren, weil er den kompletten Arbeitsspeicher des PC nutzen kann und nicht auf ein 64-KByte-Segment beschränkt ist.

Für die Verwaltung der dynamischen String-Variablen existieren verschiedene Verfahren

Eine zweite Möglichkeit besteht darin, bei jeder Änderung der Länge einer Zeichenkette die zusätzlich benötigten Zeichen in den Variablenspeicher einzufügen, indem alle nachfolgenden Variablendaten um die entsprechende Anzahl von Bytes verschoben werden. Zwar wäre dies sicher das speicherwirtschaftlichste Verfahren zum Ablegen von String-Variablen, allerdings würde es dieses Verfahren erfordern, alle Zeiger auf Zeichenketten, die hinter der geänderten Zeichenkette stehen, ebenfalls umzuändern, weil sich die zu diesen Zeigern gehörenden Zeichenketten ebenfalls im Speicher verschieben. Dadurch würden bei jeder Neu Zuweisung einer Zeichenkette unter Um-

ständen sehr viele Rechenschritte zur Neuberechnung der Zeiger notwendig – die Verarbeitungsgeschwindigkeit eines Programms würde dadurch stark leiden.

Der GW-Basic-Interpreter verwendet aufgrund der aufgeführten Nachteile keine der beiden genannten Methoden. Zeichenketten werden vielmehr so verwaltet, daß bei jeder Neuzuweisung, die die Länge einer String-Variablen ändert (zum Beispiel durch Zuweisungen der Form »n\$ = n\$ + k\$«), der neue Variableninhalt wie eine ganz neue Variable am Ende des String-Speicherbereichs abgelegt wird.

Bei jeder Neuzuweisung einer String-Variablen wird folglich der alte Zeiger auf diese Variable gelöscht und statt dessen die Variable an das Ende des String-Speichers angehängt. Das Problem bei dieser Methode liegt darin, daß auf diese Weise während des Ablaufs eines Basic-Programms (speziell bei intensiven String-Operationen) eine Vielzahl von »String-Leichen« entstehen. Darunter versteht man Zeichenketten, deren Inhalt zwar nicht mehr benötigt wird, deren Speicherplatz aber trotzdem nicht frei zur Verfügung steht.

Von Zeit zu Zeit beseitigt der GW-Basic-Interpreter die String-Leichen aus seinem Speicher

Damit der komplette, GW-Basic zur Verfügung stehende Speicher durch den String-Speicherbereich nicht zu schnell belegt ist, löscht der GW-Basic-Interpreter von Zeit zu Zeit all diese String-Leichen aus dem Speicher und gibt auf diese Weise wieder wertvollen Speicherbereich frei. Diesen Vorgang der Freigabe von einstmals belegtem Speicherbereich bezeichnet man mit der bereits erwähnten Garbage Collection. Dabei werden die unbenutzten Zeichenketten gelöscht und die benutzten Zeichenketten so im String-Speicher verschoben, daß zwischen ihnen keine Spei-

cherblöcke mehr frei sind. Dazu müssen alle Zeiger auf die Zeichenketten (zeitintensiv) an die neuen Speicheradressen angepaßt werden. Nach jeder Garbage Collection ändern sich also die Adressen der meisten Zeichenketten.

Über die Funktion »varptr« können Sie in GW-Basic direkt auf den Inhalt von Variablen zugreifen

Auf die Speicheradressen der numerischen und alphanumerischen Variablen haben Sie über die Basic-Funktion »varptr()« und »varptr\$()« Zugriff. Diese Funktionen erlauben die Feststellung der Speicheradresse einer beliebigen Variablen. Die Syntax der beiden Funktionen lautet

```
adresse=varptr(variable)
adresse=varptr$(variable)
```

wobei mit »variable« die Variable, deren Speicheradresse ermittelt werden soll, und mit »adresse« das Ergebnis der Varptr-Funktion bezeichnet ist.

Während die Funktion »varptr()« die Adresse des Inhalts für jede Variable ermittelt, liest die Funktion »varptr\$()« lediglich den internen Variablenzeiger von GW-Basic aus. Um den Unterschied zwischen diesen beiden Funktionen zu verstehen, müssen Sie sich mit der Methodik des Interpreters beim Speichern von Variablen befassen.

Die Variablen für ein GW-Basic-Programm werden in einer Liste im Datenbereich festgehalten. Diese Liste enthält für jede im Programm verwendete oder einmal definierte Variable einen Eintrag. Das erste Zeichen eines solchen Eintrags besteht aus der Kennung des Variablentyps, die in Tabelle 1 für alle Variablentypen aufgeführt ist. Nach der Kennung folgen die beiden ersten Zeichen des Variablennamens im ASCII-Format. Nach diesen beiden Zeichen folgt ein Wert, der die Anzahl der noch verbleibenden Zeichen des Variablennamens angibt. Die ASCII-

Werte dieser verbleibenden Buchstaben folgen in den nächsten Speicherzellen. Darauf folgt dann der Inhalt der Variablen selbst. Tabelle 2 zeigt den Aufbau eines Eintrags in der Variablenliste.

Der Aufbau der Variablenliste nach den beschriebenen Gesetzmäßigkeiten bietet für den GW-Basic-Interpreter große Vorteile bei der Verarbeitung der Variablen. Zunächst vergleicht er bei der Suche nach einem Variablenwert in der Variablenliste nur die ersten beiden Zeichen des Variablennamens mit den Listeneinträgen. Erst wenn diese beiden Zeichen eine Übereinstimmung mit den Variablennamen ergeben, werden auch die verbleibenden Zeichen des Listeneintrags überprüft. Dadurch spart sich der Interpreter unnötige Vergleichsoperationen mit langen Zeichenketten, wenn schon an den ersten beiden Zeichen abgelesen werden kann, daß der Name nicht mit dem gesuchten übereinstimmen kann.

Der Eintrag für eine String-Variable enthält anstelle der zugehörigen Zeichenkette einen Descriptor

Besonders interessant an den Einträgen in die Variablenliste sind die Bytes, in denen der Inhalt der Variablen abgelegt wird. Hier tritt das schon beschriebene Problem der unterschiedlichen Verwaltung von statischen und dynamischen Variablen auf. Bei den numerischen Variablen, die immer statisch sind, kann der Inhalt der Variablen unmittelbar hinter den Variablennamen gespeichert werden, denn sie belegen eine konstante Länge.

Anders sieht es mit den String-Variablen aus. Weil sie sich in der Länge dynamisch ändern können, kann der Interpreter ihren Inhalt nicht unmittelbar hinter dem entsprechenden Variablennamen abspeichern. Anstelle der zwei, vier oder acht Byte Speicherplatz, die die numerischen Varia-

Typ	Symbol	Länge/interne Kennung
Zeichenkette (String)	\$	3
Ganzzahl (Integer)	%	2
einfachgenaue Fließkommazahl (single real)	! oder keine Kennung	4
doppeltgenaue Fließkommazahl (double real)	#	8

Tabelle 1. Die unterschiedlichen Variablentypen von GW-Basic

Byte (hex)	Inhalt
00	Variablen-Kennung (siehe Tabelle 1)
01-02	die ersten zwei Zeichen des Variablennamens im ASCII-Format
03	Anzahl der verbleibenden Zeichen des Variablennamens
04- ..	verbleibende Zeichen des Variablennamens im ASCII-Format
.. - ..	Inhalt der Variablen (bei numerischen Variablen unmittelbar, bei String-Variablen Zeiger auf Zeichenkette)

Tabelle 2. Der Aufbau eines Variableneintrags

blen für ihren Inhalt benötigen, trägt der Interpreter bei String-Variablen deshalb einen drei Byte langen Zeiger auf den eigentlichen Speicherbereich der Zeichenkette ein. Man nennt diesen Zeiger den »String Descriptor« oder nur kurz »Descriptor«. Im ersten Byte des Descriptors ist die Länge der Zeichenkette gespeichert, in den verbleibenden zwei Byte ist deren Adresse im String-Speicherbereich abgelegt.

Anhand dieser Ausführungen läßt sich nun sehr gut die Bedeutung des Unterschiedes zwischen den Funktionen »varptr()« und »varptr\$()« erklären.

»varptr\$()« liefert eine Zeichenkette zurück, die aus den zu einer Variablen gehörenden Bytes der Variablenliste besteht. Das erste Byte ist die interne Kennung der Variablen, und dahinter folgt die Adresse der Variablen in der Variablenliste. Nur bei numerischen Variablen ist diese Adresse identisch mit der Speicheradresse des Variableninhalts; bei String-Variablen erhalten Sie statt dessen einen Zeiger auf den Descriptor – nicht auf den Datenbereich. Demgegenüber ermittelt die Funktion »varptr()« immer die Adresse des Datenbereichs einer gegebenen Variablen.

Das Format für den Inhalt von numerischen Variablen ist zum Teil recht trickreich

Wie Sie Tabelle 1 entnehmen können, besitzen die GW-Basic-Variablentypen abgesehen von den String-Variablen, die je nach Länge unterschiedlich viel Speicherplatz belegen, ein konstantes Speicherformat. Bei den Ganzzahlen mit zwei Byte Länge ist das Format noch relativ leicht verständlich: Hier wird der Variablenwert im 16-Bit-Binärformat auf zwei Byte aufgeteilt. Das erste Byte enthält den niederwertigen Teil des Variablenwertes, das zweite Byte den höherwertigen Teil. Wenn Sie also mit

```
wert%=100
```

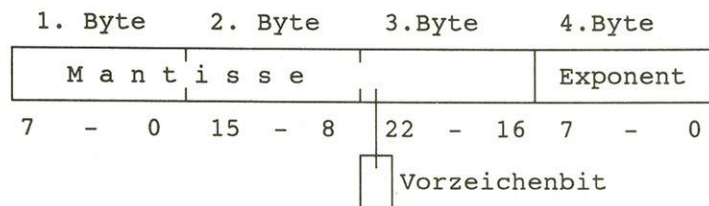
eine ganzzahlige Variable mit dem Wert 100 definieren, dann können Sie diesen Wert mit der Befehlszeile

```
print
peek(varptr(wert%))+256*peek(varptr
(wert%))
```

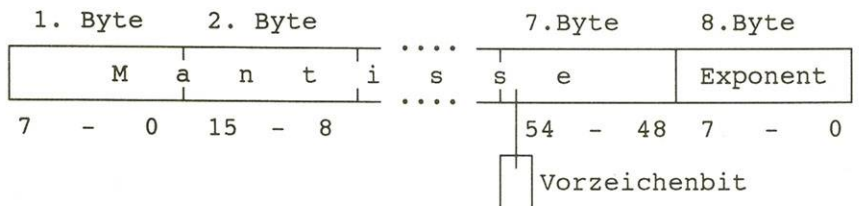
wieder aus dem Variablenspeicher auslesen.

Auch für Zeichenketten ist das Speicherformat einsichtig. Hier werden die einzelnen Zeichen des Strings unmittelbar hintereinander im Speicher abgelegt. Die Länge und Adresse dieser Zeichenkette ist aus dem

Format einer einfachgenauen Fließkommazahl:



Format einer doppelgenauen Fließkommazahl:



Das binäre Speicherformat für einfach- und doppelgenaue Fließkommazahlen

String Descriptor in der Variablenliste ersichtlich.

Das Format von Integer- und String-Variablen ist damit klar. Doch wie werden einfachgenaue Fließkommazahlen in nur vier und doppelgenaue in nur acht Byte untergebracht? Sie müssen sich dabei vor Augen halten, daß zum Beispiel die Zahl 0,0000220567 eine gültige Fließkommazahl einfacher Genauigkeit und die Zahl 267844007860000000 eine gültige Zahl doppelter Genauigkeit ist.

Einfach- und doppelgenaue Zahlen werden im sogenannten Fließkommaformat abgelegt

Der Trick zur Speicherung dieser Zahlen liegt im sogenannten »Fließkommaformat«. Das ist eine Methode, die beliebige Kommazahlen auf pfiffige Weise in wenigen Informationseinheiten (also Bits) unterbringt. Sicher wird den meisten Lesern die wissenschaftliche Notation beim Umgang mit Zahlen bekannt sein. Dabei zerlegt man eine gegebene Zahl, indem man sie durch die höchste erhaltene Potenz der Basis des Zahlensystems (in der Regel das Dezimalsystem) dividiert. So lassen sich alle Zahlen größer 1 und kleiner -1 eines gegebenen Zahlensystems darstellen. Hierzu einige Beispiele:

```
400 = 4 x 102
33 = 3,3 x 101
-7340 = -7,34 x 103
```

Dehnt man die Exponenten für die Basis 10 auch auf den negativen Bereich aus, dann lassen sich auch die Zahlen zwischen 1 und -1 verewenden:

```
0,76 = 7,6 x 10-1
0,004 = 4 x 10-3
```

In der Exponentendarstellung von GW-Basic werden diese Zahlen auch als »7,6E-1« (für 0,76) oder »4E-3« (für 0,004) dargestellt. Der Buchstabe »E« weist darauf hin, daß es sich bei dem folgenden Wert um den Exponenten handelt.

In der normalisierten Darstellung wird eine Zahl als Mantisse mit einer Vorkommate und einem Exponenten angezeigt

In der Exponentendarstellung ist eine beliebige Zahl eindeutig durch die sogenannte Mantisse oder den Signifikanten (das ist der Wert vor dem »E«) und den Exponenten (das ist der Wert hinter dem »E«) gekennzeichnet, sofern die Basis des Zahlensystems (in diesem Fall 10) bekannt ist. Bei einer gegebenen Mantisse von 4 und einem Exponenten von -3 weiß man also, daß es sich um die Zahl 0,004 handeln muß.

In der Exponentendarstellung mit nur einer Stelle vor dem Komma, die auch als »normalisierte Darstellung« bezeichnet wird, lassen sich Rechenoperationen sehr einfach durchführen. Die Multiplikation von -0,76 und

0,004 wird zur Multiplikation von $-7,6E-1$ und $4E-3$. Hier können nun einfach die Mantissen multipliziert und die Exponenten addiert werden. Das Ergebnis lautet also $-30,4E-4$ oder in normalisierter Darstellung $-3,04E-3$ ($-0,00304$).

Nun werden in einem Computer aber die Zahlen nicht im Dezimalformat, sondern in binärer Darstellung gespeichert. Anstelle von Dezimalzahlen liegen die Zahlen deshalb als Mantisse und Exponent im Binärformat vor. Dabei bezieht sich der Exponent dann ebenfalls nicht mehr auf das Dezimal-, sondern auf das Binärsystem. Um diese Darstellung zu verdeutlichen, folgt ein Schema, das aufweist, wie sich eine binäre Fließkommazahl ($1,011E1$) in das Dezimalsystem umwandeln läßt:

$$\begin{array}{r} 1 \times 2^1 = 2 \\ + 0 \times 2^0 = 0 \\ + 1 \times 2^{-1} = 0,5 \\ + 1 \times 2^{-2} = 0,25 \\ \hline 2,75 \end{array}$$

Wie sieht es bei der binären Exponentendarstellung aber mit dem Speichern von negativen Zahlen oder Zahlen kleiner als 1 aus? Für den ersten Fall muß die Mantisse negativ werden, für den zweiten Fall muß der Exponent einen negativen Wert erhalten. Und wie stellt man negative Binärzahlen dar? Die einfachste Methode ist ein separates Bit, dessen Zustand darüber Aufschluß gibt, ob die Zahl negativ oder positiv ist. Sowohl für die Mantisse als auch für den Exponenten muß deshalb ein zusätzliches Bit reserviert werden. Im Gegensatz zu anderen Verfahren (zum Beispiel dem Zweierkomplement) kommt man bei dieser Vorzeichen-Betragsdarstellung ohne großes Umrechnen zurecht, wenngleich durch dieses Verfahren bei positiven Zahlen ein Bit ungenutzt bleibt.

GW-Basic speichert den Inhalt von Fließkommazahlen wahlweise 32 oder 64 Bit breit ab

Die Genauigkeit einer Fließkommazahl hängt davon ab, wie viele Bits für die Mantisse und den Exponenten zur Verfügung stehen. In GW-Basic werden die Fließkommazahlen in vier Byte (einfache Genauigkeit) oder acht Byte (doppelte Genauigkeit) gespeichert. Es stehen also 32 beziehungsweise 64 Bit für einen Fließkommawert zur Verfügung.

Bei einfachgenauen Fließkommazahlen werden in GW-Basic 24 Bit für die Mantisse und 8 Bit für den Exponenten und bei doppeltgenauen Fließkommazahlen 56 Bit für die Mantisse und wiederum 8 Bit für den

Funktion	Variablentyp	Länge der Zeichenkette
mki\$()	Ganzzahl (Integer)	2
mks\$()	einfachgenau (single real)	4
mkd\$()	doppeltgenau (double real)	8

Tabelle 3. Die drei Funktionen zur Datenkonvertierung

Exponenten reserviert. Dadurch ergibt sich für einfachgenaue Fließkommazahlen ein Wertebereich von $-1,701412E38$ bis $1,701412E38$ und für doppeltgenaue Fließkommazahlen ein Wertebereich von $-1,701411834604692E38$ bis $1,701411834604692E38$.

Während viele Programmiersprachen den Exponenten wie die Mantisse mit einem eigenen Vorzeichenbit speichern, benutzt der GW-Basic-Interpreter ein anderes Verfahren. GW-Basic stellt den Exponenten nämlich jeweils mit einem Offset von 81hex dar, das heißt, zum binären Exponenten einer Zahl wird der Wert 81hex addiert, so daß auf diese Weise ebenfalls negative Werte zugelassen sind. Dieses Verfahren bezeichnet man auch als Komplementdarstellung.

Die Mantisse einer Fließkommazahl ist in der Form gespeichert, daß das niederwertigste Byte zuerst und das höchstwertige Byte zuletzt folgt. Weil bei der normalisierten Darstellung im Binärsystem immer die Ziffer 1 vor dem Komma steht, das heißt Bit 7 des höchstwertigen Bit der Mantisse immer gesetzt wäre, wird dieses Bit statt dessen zur Darstellung des Mantisse-Vorzeichens benutzt.

Die Darstellung von Fließkommazahlen in GW-Basic ist sehr trickreich

Soll der GW-Basic-Interpreter beispielsweise den Wert 1,25 im Fließkommamodus darstellen, so wird dieser intern binär als $1,01 \times 2^0$ gespeichert. Weil die 1 vor dem Komma jedoch grundsätzlich vorhanden ist, wird sie implizit vorausgesetzt und nur die Mantisse 01 gespeichert. Der Interpreter weiß jedoch, daß jede gespeicherte Ziffernfolge noch von einer 1 und dem Komma angeführt wird.

In der kompletten Fließkommadarstellung wird die Mantisse des dezimalen Wertes 1,25 also mit den 24 Bit 001000000000000000000000 dargestellt. (Das zusätzliche Null-Bit zu Beginn definiert das Vorzeichen.) Weil der GW-Basic-Interpreter die Ziffernfolge in Gruppen zu je acht Bit jedoch verkehrt herum im Variablenspeicher ablegt (niederwertiges Byte

vor höherwertigem Byte), ergibt sich damit die Reihenfolge

00000000 00000000 00100000

Dabei stellt das höchste Bit der dritten 8-Bit-Gruppe das Vorzeichen dar. Nach dem gleichen Schema werden auch die Mantissen von doppeltgenauen Fließkommazahlen gespeichert, nur werden in diesem Fall sieben statt drei Byte zur Mantissendarstellung herangezogen, so daß die Genauigkeit dadurch wesentlich höher ist.

Das vierte beziehungsweise achte Byte im Speicherformat einer Fließkommazahl bildet der Exponent. Das Bild zeigt den konkreten Aufbau der einfach- und doppeltgenauen Fließkommazahlen im GW-Basic-Variablenspeicher.

Zwei kleine Programme machen das binäre Speicherformat für einfach- und doppeltgenaue Fließkommazahlen sichtbar

Mit dem Programm »sfloat.bas« in Listing 1 können Sie sich beliebige einfachgenaue Fließkommazahlen in dem Format anschauen, in dem sie von GW-Basic gespeichert werden. Lediglich die Mantissenbytes werden in die richtige Reihenfolge gedreht, damit Sie die gewohnte Darstellung der binären Werte erhalten. Das Programm zeigt die gespeicherten Zahlen im Binärformat an und setzt hinter dem Vorzeichen und vor dem Exponenten automatisch ein Freizeichen, um die Übersichtlichkeit der Darstellung zu erhöhen.

Das Programm »dfloat.bas« in Listing 2 ist das Gegenstück zu »sfloat« und zeigt das Speicherformat von doppeltgenauen Fließkommazahlen an. Um zu erklären, wie die beiden Programme funktionieren, müssen Sie die Bedeutung der Funktionen »mks\$()« und »mkd\$()« kennenlernen.

Bevor der Inhalt von Fließkomma-Variablen ausgegeben wird (zum Beispiel mit dem Print-Befehl) wird er vom GW-Basic-Interpreter immer in das gewohnte dezimale Format umgewandelt. Mit einigen Tricks kann man aber auch auf das interne Spei-

cherformat der Variablen zurückgreifen. Die erste Methode funktioniert über die Funktionen »varptr()« und »varptr\$()«. Mit ihnen kann man die Adresse einer Fließkommazahl ermitteln und ab dieser Adresse die einzelnen Bytes mit der Peek-Funktion auslesen.

Es gibt aber noch ein viel einfacheres Verfahren, bei dem Ihnen der Interpreter den Großteil der Arbeit abnimmt. Für die Speicherung von Daten in Dateien wäre es nützlich, auch lange Kommazahlen in einem kurzen Format unterbringen zu können. Zu diesem Zweck stehen unter GW-Basic die Funktionen »mki\$()«, »mks\$()« und »mkd\$()« zur Verfügung, die Ganzzahlen, einfachgenaue und doppeltgenaue Fließkommazahlen in eine zwei, vier oder acht Byte lange Zeichenkette umwandeln. Dadurch lassen sich die Zahlen auf einem Datenträger auf erheblich kleinerem Raum speichern, weil nicht für jede Stelle einer Zahl ein Byte Speicher-

platz benötigt wird. Mit den Funktionen »cvi()«, »cvs()« und »cvd()« werden die Zeichenketten nach dem Laden wieder in ihre numerische Form zurückgewandelt.

Weil die Zeichenketten, die von den Mk-Funktionen erzeugt werden, genau das Format besitzen, in dem die Daten vom GW-Basic-Interpreter intern gespeichert werden, können Sie auf diese Weise das Speicherformat wesentlich einfacher als über die Varptr-Funktionen untersuchen. Tabelle 3 zeigt, welche der Funktionen für welchen Datentyp zuständig ist und wie lang die daraus resultierende Zeichenkette ist.

Haben Sie beispielsweise mit der Zuweisung

```
doppelt# = 1.234
```

eine doppeltgenaue Fließkommazahl definiert, so enthält nach der Zuweisung

```
doppelt$ = mkd$(doppelt#)
```

die Zeichenkette »doppelt\$« die acht Byte, die der Interpreter intern zur Codierung des Formats der doppeltgenauen Fließkommazahl verwendet. Umgekehrt können Sie diese Zeichenketten mit den Cv-Funktionen wieder zurückwandeln. Die Funktion »cvi()« ist das Gegenstück zu »mki\$()«, »cvs()« das Gegenstück zu »mks\$()« und »cvd()« das Gegenstück zu »mkd\$()«.

Die Kenntnis vom internen Speicherformat für Fließkommazahlen ist sehr nützlich, denn mit diesem Wissen sind Sie beispielsweise in der Lage, bei der maschinennahen Programmierung mit GW-Basic, also dem Einbinden von Assembler Routinen oder dem Aufruf von Interrupts, auch Fließkommazahlen an Assembler-Programme zu übergeben, weiterzuverarbeiten und zurückzugeben.

(Martin Althaus/ma)

```
[6U0] 10 CLS
[PR3] 20 D(1)=3:D(2)=2:D(3)=1:D(4)=4
[GP3] 30 INPUT "Zahl? ",FLOAT#
[AY1] 40 FOR I=1 TO 4
[8L5] 50 ZAHL=ASC(MID$(MKD$(FLOAT#),D(I),1))
[243] 60 FOR J=7 TO 0 STEP -1
[GP5] 70 PRINT RIGHT$(STR$(-(2^J<=ZAHL)),1);
[NL5] 80 IF I*8-J=1 OR I*8-J=24 THEN PRINT " ";
[FB4] 90 ZAHL=ZAHL+2^J*(2^J<=ZAHL)
[8S1] 100 NEXT J,I
[5C1] 110 PRINT
[BM1] 120 GOTO 30
```

Listing 1. Das Programm »sfloat.bas« demonstriert das binäre Speicherformat für einfachgenaue Fließkommazahlen

```
[6U0] 10 CLS
[GD7] 20 D(1)=7:D(2)=6:D(3)=5:D(4)=4:D(5)=3:D(6)=2:D(7)=1:D(8)=8
[KP3] 30 INPUT "Zahl? ",FLOAT#
[UY1] 40 FOR I=1 TO 8
[VK5] 50 ZAHL=ASC(MID$(MKD$(FLOAT#),D(I),1))
[243] 60 FOR J=7 TO 0 STEP -1
[GP5] 70 PRINT RIGHT$(STR$(-(2^J<=ZAHL)),1);
[WL5] 80 IF I*8-J=1 OR I*8-J=56 THEN PRINT " ";
[FB4] 90 ZAHL=ZAHL+2^J*(2^J<=ZAHL)
[8S1] 100 NEXT J,I
[5C1] 110 PRINT
[BM1] 120 GOTO 30
```

Listing 2. »dfloat.bas« zeigt das Speicherformat für doppeltgenaue Fließkommazahlen an

Wissen Sie mehr von PCs als andere Leute?

Dann schreiben Sie uns. Die Redaktion DOS International sucht aktive Mitarbeiter, die über interessante Gebiete aus der »Computerei« berichten wollen. MS-DOS, DOS Plus, GEM, Drucker, Basteleien – es gibt nichts, worüber nicht andere Leser unserer Zeitschrift auch etwas lesen wollen.

Ihren Einsatz lassen wir uns natürlich etwas kosten. Ihr Manuskript wird selbstverständlich honoriert. Bilder, Zeichnungen, Tabellen und Listings werden gesondert berechnet.

Doch bevor Sie sich hinsetzen und das Schreiben anfangen: Rufen Sie uns an. Sagen Sie uns, was Sie schreiben wollen, und wir klären ob – und wann – der Artikel in unser Konzept paßt.

Der Artikel muß mit einem Textsystem erfaßt werden. Wir brauchen von Ihnen eine Diskette im MS-DOS-Format, auf der der Text als reines ASCII-Feld steht.

Schreiben Sie uns auch, wenn Sie nur Vorschläge für Themen haben. DOS International ist nämlich eine Zeitschrift von aktiven PC-Besitzern für aktive PC-Benutzer. Besonders das DOS-Forum lebt von Ihrer Mitarbeit. Auch die Redaktion kann nicht alle Antworten wissen. Fragen an uns, die nicht sofort beantwortet werden können, werden im DOS-Forum veröffentlicht. Wissen Sie eine Antwort, so wird diese veröffentlicht.

Sie sehen: DOS International ist die PC-Zeitschrift, in der das Wissen der PC-Benutzer untereinander ausgetauscht wird.

Redaktion · DOS International · Hans-Stießberger-Str. 3 · 8013 Haar/München · Tel.: (089) 46 10 01-0

Basic-Programm selbstgemacht oder Über das interne Format von Basic-Programmen

Wer versucht, ein Basic-Programm in einen Texteditor zu laden, wird feststellen, daß das Programm nicht als echte Textdatei auf dem Datenträger gespeichert, sondern vielmehr in einer besonderen Art und Weise, nämlich in codierter Form als sogenannte Token abgelegt wird.

Die Codierung von Basic-Befehlen in Programmen, die sogenannte Token-Schreibweise, hat zwei Gründe: Einerseits wird so wertvoller Speicherplatz gespart, da die codierten Befehle erheblich weniger Platz beanspruchen als die Befehle im ASCII-Format, und andererseits beschleunigt diese Methode die Ausführung eines Programms, da die Programmzeilen nicht erst nach Schlüsselwörtern untersucht werden müssen.

Die Kenntnis dieser internen Programmcodierung ist vor allem aus folgenden Gründen interessant: Zum einen kann man damit sehr leistungsfähige Anwendungen realisieren, zum Beispiel die Erzeugung eines Basic-Programms durch ein anderes Programm. Zum anderen führt die Kenntnis der internen Codierung aber auch zu einem fundamentalen Verständnis der Arbeitsweise des Basic-Interpreters und seiner Leistungsgrenzen.

Wie kommt es überhaupt zu dieser Codierung? Sobald im Basic-Editor eine Zeile eingegeben wird, wertet der Interpreter diese aus. Als erstes stellt er fest, ob die ersten Zeichen der Zeile eine gültige Zeilennummer darstellen. Ist das nicht der Fall, wird die Zeile als Direkteingabe verstanden. Direkteingaben sind Kommandos wie das folgende, zu denen der Interpreter unmittelbar ein Ergebnis liefern kann.

```
print 6*(143*4766)
```

Der Interpreter versucht bei diesen Direkteingaben, die Kommandos in der Zeile direkt auszuführen. Läßt sich die Direkteingabe nicht eindeutig auswerten, meldet er »Syntax Error«.

Waren hingegen die ersten Zeichen der Zeile gültige Bestandteile einer Zeilennummer, betrachtet der Interpreter die gesamte Zeile als Programmzeile und macht sich daran, diese in das Programm im Speicher einzuordnen. Der erste Schritt dazu ist das grammatische Zergliedern der Zeile. Bei diesem Vorgang wird eine Zeile auf Schlüsselwörter untersucht; hier beginnt die Codierung des Programms. Jedes gültige Basic-Kommando, das der Interpreter im Quell-

text findet, wird durch ein »Token« ersetzt. Jedes dieser Token ist ein oder zwei Byte lang. So hat zum Beispiel der Print-Befehl das Token 91hex und die Val-Funktion das Token FF94hex. Die Tabellen 1 und 2 fassen die Ein- und Zwei-Byte-Token zusammen.

Nach der Eingabe einer Programmzeile wandelt der Interpreter diese in Token um

Auf diese Weise ersetzt der Interpreter alle Schlüsselwörter im Programmtext durch Token. Im gleichen Schritt werden auch alle im Programm vorkommenden unmittelbaren Werte gegen ihre binäre Entsprechung ausgetauscht. In der Programmzeile

```
if a < 120 then end
```

wird »120« also durch die Hexadezimalzahl 78hex ersetzt. Damit der Interpreter erkennt, in welchem Format die Zahl vorliegt, wird vor dem eigentlichen Wert ein Code in den Programmtext eingefügt, der das nach-

folgende Zahlenformat beschreibt. Ein Code von 0Ehex steht für eine folgende Zwei-Byte-Zahl, ein Code von 0Fhex für eine folgende Ein-Byte-Zahl. Zeichenketten im Quelltext werden nicht umgewandelt. Sie bleiben im Klartext zwischen zwei Anführungszeichen stehen.

Neben der Umwandlung der Basic-Schlüsselwörter in Token und der Umwandlung der Variablenwerte in ihre codierte Form nimmt sich der Interpreter dann auch noch die Zeilennummer vor. Im Konzept des Basic-Interpreters sind Zeilennummern unumgänglich. Denn wie wäre es möglich, ohne Zeilennummern neue Programmzeilen in den bereits codierten Quelltext einzusortieren? Für die Zeilennummer sind generell zwei Byte reserviert. Sie wird also als 16-Bit-Integer-Wert im Programmtext codiert. Aus dieser Beschränkung auf 16 Bit ergibt sich auch die in Basic höchste Zeilennummer von 65535, weil das die größte mit 16 Bit darstellbare Zahl ist. Eine Null kennzeichnet das Ende jeder Programmzeile.

Nun wäre es ja für den Interpreter eine sehr aufwendige Aufgabe zu untersuchen, ob eine Zeilennummer bereits vorhanden oder wo eine neue Zeile in den Programmtext einzuordnen ist. Dazu müßte er jede Zeile bis zu ihrem Ende verfolgen, um die nächste Zeilennummer auszulesen. So würde sich der Interpreter so lange durch den Quelltext »hangeln«, bis er die richtige Zeilennummer gefunden hätte. Da diese Methode zu zeitaufwendig ist, trägt der Interpreter vor jeder Zeilennummer noch ein,

```
FF FE 0A 0A 00 C0 00 18 0B 14 00 85 22 44 61 74
65 69 6E 61 6D 65 3F 20 22 3B 44 4E 41 4D 45 24
00 30 0B 1E 00 BA 20 44 4E 41 4D 45 24 20 41 53
20 23 12 20 FF 92 E7 12 00 44 0B 28 00 FE 82 23
12 2C 12 20 41 53 20 42 59 54 45 24 00 52 0B 32
00 49 25 E7 11 3A 4A 25 E7 12 00 64 0B 3C 00 4C
41 45 4E 47 45 25 E7 FF A5 28 12 29 00 6A 0B 46
00 C0 00 7D 0B 50 00 B1 E9 20 49 25 E8 E7 4C 41
45 4E 47 45 25 00 86 0B 5A 00 FE 89 23 12 00 9F
0B 64 00 CA 20 4A 25 2C 28 28 49 25 20 F3 20 0F
10 29 EB 14 E9 12 29 00 C2 0B 6E 00 91 20 FF 82
28 22 30 22 E9 FF 9A 28 FF 95 28 42 59 54 45 24
29 29 2C 13 29 3B 22 20 22 3B 00 DA 0B 78 00 CA
20 4A 25 2C 0F 3C E9 28 49 25 20 F3 20 0F 10 E9
12 29 00 FF 0B 82 00 8B 20 42 59 54 45 24 E8 22
20 22 20 CD 20 91 20 22 2E 22 3B 20 3A A1 20 91
20 42 59 54 45 24 3B 00 13 0C 8C 00 8B 20 DE E7
22 20 22 20 CD 20 8D 20 0E 04 01 00 1F 0C 96 00
49 25 E7 49 25 E9 12 00 3B 0C A0 00 8B 20 28 49
25 20 F3 20 0F 10 29 E7 11 20 CD 20 4A 25 E7 4A
25 E9 12 00 4D 0C AA 00 8B 20 4A 25 E8 0F 16 20
CD 20 0E E6 00 00 59 0C B4 00 CA 20 0F 17 2C 0F
19 00 7A 0C BE 00 91 20 22 42 69 74 74 65 20 4C
65 65 72 74 61 73 74 65 20 64 72 81 63 6B 65 6E
22 3B 00 84 0C C8 00 8D 20 0E 04 01 00 8A 0C D2
00 C0 00 93 0C DC 00 4A 25 E7 12 00 99 0C E6 0E
B2 00 A1 0C F0 00 BB 20 12 00 A7 0C FA 00 81 00
B5 0C 04 01 B1 E9 20 DE E8 E6 22 20 22 00 BB 0C
0E 01 B2 00 C1 0C 18 01 E8 00 00 00 1A 00
```

```
...L.... "Dat
einame?";DNAMES
.0...| DNAMES AS
#. ...D.(.# #
... AS BYTES.R.2
I% .J% .d.<.L
AENGEX% (.).J.F
.L).P. I% LA
ENGEX% .Z.#...
.d. J%,(I% < .
.) .).T.n.
("0" U( (BYTES
)),.);" ";.r.x. J%
J%,< (I% < ..
). ... BYTES"
" = ".; :
BYTES;... |
" = .....
I% I% .;.. (I
% < ..) . - J% J
% .M.. J%..
- ...Y..J%....
..z.J. "Bitte L
eertaste drücken
.;.L. .... T
.L. J% .Ö.
..J% ..u.
J% J% " "J.
..J%..X....
```

So stellt »seetoken.bas« Basic-Programme dar

an welcher Speicherstelle die nächste Programmzeile beginnt.

Diese Speicherstelle stellt den Offset innerhalb des aktuellen Segments dar, in dem der Interpreter alle Daten, also den Programmtext und die Variablen, ablegt. Der Anfang des Programmtextes hat dabei die relative Adresse 0BF7hex. Ein Zeiger mit dem Wert Null macht kenntlich, daß keine weitere Programmzeile im Quelltext mehr folgt. Hinter diesem Zeiger sind aber dann noch die Codes 1Ahex und Null abzulegen, die das Ende eines Programms festlegen. Das hat zur Folge, daß selbst bei einem Programm von nur einer Zeile mindestens ein gültiger Zeiger auf die nächste Programmzeile im Programmtext enthalten sein muß. Erst an dieser Stelle wird dann das Ende des Programms durch einen Zeiger von Null und die beiden folgenden Bytes 1Ahex und Null markiert. Das allgemeine Format einer Programmzeile im Token-Format lautet also

<Zeiger> <Zeilennummer> <Token-
Zeile> <Null>

Unmittelbar darauf folgt die nächste Zeile, wieder zuerst mit dem Zeiger auf die übernächste Programmzeile, dann mit der Zwei-Byte-Zeilennummer und der Programmzeile in Token. Vor jedem Programm legt der In-

terpreter außerdem einen ein Byte langen Code in Form des Wertes FFhex ab. Daran kann er erkennen, daß das Programm schon in der Token-Form vorliegt und nicht erst umgewandelt werden muß.

Wird zum Beispiel ein Programm mit dem Parameter »a« als ASCII-Datei abgespeichert, erkennt der Interpreter nach dem Laden sofort, daß er es erst in die Token-Form umwandeln muß. Direkt auf dieses Kennzeichen folgt die erste Programmzeile des Quelltextes, eingeleitet durch den Zeiger auf die nächste Zeile.

Mit »Seetoken« kommen Sie hinter die Geheimnisse Ihres Basic-Interpreters

Es ist sehr reizvoll, sich Programme im Token-Format anzeigen zu lassen. Dazu dient das kleine Basic-Programm »seetoken.bas«. Nach dem Start des Programms wird der Name eines Basic-Programms eingegeben, das auf dem Datenträger in der Token-Form gespeichert ist. Seetoken stellt die Datei dann auf dem Bildschirm sowohl in ihrer ASCII-Darstellung als auch in der hexadezimalen Schreibweise dar. So lassen sich Programme sehr genau analysieren. Ist der Bildschirm voll, fordert Seetoken

automatisch dazu auf, die Leertaste zu drücken, worauf in der gezeigten Datei eine Bildschirmseite weitergescrollt wird. Des weiteren läßt sich mit der Leertaste der Listvorgang anhalten, durch erneutes Drücken wird damit fortgefahren.

Bild 1 zeigt einen Lauf mit Seetoken, bei dem als Dateiname der eigene Programmname eingegeben wurde. Hier als Beispiel die Zeile 140 aus dem Quelltext:

```
140 if inkey$="" then gosub 260
```

Diese Zeile sorgt dafür, daß beim Drücken der Leertaste in eine Warteschleife verzweigt wird. Im Token-Format des Programms wird diese Zeile durch die Zahlenfolge

```
13 0C 8C 00 8B 20 DE E7 22 20  
22 20 CD 20 8D 20 0E 04 01 00
```

repräsentiert. Diese Zeichenfolge ist erheblich kürzer als die gesamte Programmzeile im ASCII-Format. Mit der Kenntnis des Aufbaus einzelner Programmzeilen läßt sich diese Zeile nun genau untersuchen. Sie beginnt mit den Werten 13hex und 0Chex, die als Zwei-Byte-Zeiger auf die nächste Programmzeile dienen. Die nächste Programmzeile beginnt also beim Offset 0C13hex im aktuellen Segment. Dem Zeiger folgt die Zeilennummer der Programmzeile. Sie ist in der Reihenfolge niederwertiges Byte vor höherwertigem Byte abgelegt, lautet also 008Chex. Sie stellt die hexadezimale Schreibweise von 140 dar (8x16 + 12 = 140). Danach folgt das erste Token. Wie Sie aus der

Token	Befehl	Token	Befehl	Token	Befehl
81	end	A6	edit	CA	locate
82	for	A7	error	CC	to
83	next	A8	resume	CD	then
84	data	A9	delete	CE	tab
85	input	AA	auto	CF	step
86	dim	AB	renum	D0	usr
87	read	AC	defstr	D1	fn
88	let	AD	defint	D2	spc
89	goto	AE	defsgn	D3	not
8A	run	AF	defdbl	D4	erl
8B	if	B0	line	D5	err
8C	restore	B1	while	D6	string\$
8D	gosub	B2	wend	D7	using
8E	return	B3	call	DA	varptr
8F	rem	B7	write	DB	crsln
90	stop	B8	option	DC	point
91	print	B9	random-	DD	off
92	clear		mize	DE	inkey\$
93	list	BA	open	E6	>
94	new	BB	close	E7	=
95	on	BC	load	E8	<
96	wait	BD	merge	E9	+
97	def	BE	save	EA	-
98	poke	BF	color	EB	*
99	cont	C0	cls	EC	/
9C	out	C1	motor	ED	↑
9D	lprint	C2	bsave	EE	and
9E	llist	C3	bload	EF	or
A0	width	C4	sound	F0	xor
A1	else	C5	beep	F1	eqv
A2	tron	C6	pset	F2	imd
A3	troff	C7	preset	F3	mod
A4	swap	C8	screen	FF	pen
A5	erase	C9	key		

Tabelle 1. Die Basic-Token mit einem Byte Länge

Token	Befehl	Token	Befehl	Token	Befehl
FE01	cvi	FE95	erdev	FF8C	cos
FE02	cvs	FE96	ioctl	FF8D	tan
FE03	cvd	FE97	chdir	FF8E	atn
FE04	mkis\$	FE98	mkdir	FF8F	fre
FE05	mks\$	FE99	rmdir	FF90	inp
FE06	mkd\$	FE9A	shell	FF91	pos
FE0B	exterr	FE9B	environ	FF92	len
FE01	files	FE9C	view	FF93	str\$
FE02	field	FE9D	window	FF94	val
FE03	system	FE9E	pmap	FF95	asc
FE04	name	FE9F	palette	FF96	chr\$
FE05	lset	FEA0	lcopy	FF97	peek
FE06	rset	FEA1	calls	FF98	space\$
FE07	kill	FEA5	pcopy	FF99	oct\$
FE08	put	FEA7	lock	FF9A	hex\$
FE09	get	FEA8	unlock	FF9B	lpos
FE0A	reset	FF81	left\$	FF9C	cint
FE0B	common	FF82	right\$	FF9D	csng
FE0C	chain	FF83	mid\$	FF9E	cdbl
FE0D	date\$	FF84	sgn	FF9F	fix
FE0E	time\$	FF85	int	FFA0	pen
FE0F	paint	FF86	abs	FFA1	stick
FE10	com	FF87	sqr	FFA2	strig
FE11	circle	FF88	rnd	FFA3	eof
FE12	draw	FF89	sin	FFA4	loc
FE13	play	FF8A	log	FFA5	lof
FE14	timer	FF8B	exp		

Tabelle 2. Die Basic-Token mit zwei Byte Länge

Tabelle entnehmen, stellt 8Bhex das Token für den If-Befehl dar. Der nächste Code, 20hex, ist der Code des Leerzeichens hinter dem »if«.

Um dem Programmierer die Möglichkeit zu geben, Programmzeilen oder Teile davon optisch einzurücken, werden Leerzeichen zwischen Befehlen oder zwischen Befehlen und ihren Argumenten komplett in die Token-Codierung übernommen. Nach dem Leerzeichen folgt das zweite Token, der Code DEhex für die Inkey\$-Funktion. Dahinter steht das Token E7hex für das Gleichheitszeichen. Wie beschrieben, werden Zeichenketten im Programm nicht umgewandelt, sondern inklusive der Anführungszeichen am Beginn und am Ende in das Token-Format übernommen. Hinter dem Gleichheitszeichen finden Sie also die drei Codes 22hex, 20hex und 22hex, die ein in Anführungszeichen eingeschlossenes Leerzeichen beschreiben. Dahinter finden Sie erneut den Wert 20hex für das Leerzeichen vor dem Then-Befehl. Dieser Befehl ist als nächster in seiner Token-Schreibweise zu finden. Wie aus der Tabelle ersichtlich, wird er durch den Wert CDhex codiert. Auch auf diesem Befehl folgt wieder ein Leerzeichen (20hex). Als nächstes erscheint im Token-Programm der Code 8Dhex. Das ist das Token für den Gosub-Befehl. Hinter ihm wird mit dem Code 20hex wieder ein Leerzeichen gespeichert.

An den folgenden drei Byte läßt sich sehr gut erkennen, wie der Interpreter Zahlen, die als ASCII-Sequenz eingegeben wurden, intern in Binärzahlen umwandelt. Zuerst folgt auf das Leerzeichen hinter dem Gosub-Kommando der Code 0Ehex. Dieses Zeichen zeigt an, daß die nächsten beiden Werte als zusammenhängende Zwei-Byte-Zahl zu verstehen sind. Diese Zahl ist das Sprungziel des Gosub-Befehls. Im aktuellen Beispiel finden Sie hier 04hex und 01hex, also die Zwei-Byte-Zahl 0104hex, die aber dezimal 260 darstellt

($1 \times 256 + 4 = 260$) und damit genau auf die Zeilennummer des Unterprogramms im Programmtext weist. Das Ende dieser Programmzeile kennzeichnet die nachfolgende Null. Auf diese Weise wandelt der Interpreter das gesamte Basic-Programm in ein komprimiertes Token-Format um. Für das Laufzeitverhalten des Programms bietet das enorme Vorteile. Bei der Ausführung eines Programms muß nicht mehr jede Basic-Zeile nach bestimmten Schlüsselwörtern durchsucht werden. Vielmehr ist der Interpreter in der Lage, bereits an einem oder zwei Zeichen (so lang sind die Token maximal) zu erkennen, um welchen Befehl es sich handelt. Anhand dieses Codes kann er dann sofort ein Unterprogramm aufrufen, das die Ausführung des entsprechenden Befehls bewirkt.

Wer den Aufbau eines Basic-Programms in Token-Schreibweise kennt, kann Programme schreiben, die selbst Programme schreiben

Mit der Kenntnis dieser internen Codierung der Basic-Programme lassen sich sehr leistungsfähige Anwendungen programmieren. Sie können relativ einfache Programme direkt auf einem Datenträger generieren. Dieses Verfahren finden Sie anhand einer kurzen Anwendung einmal durchgespielt.

Auf der Diskette soll ein Programm erzeugt werden, das nur die Befehlszeile

10 cls

enthält. Wie wird nun solch ein Programm erzeugt? Die – für ein so einfaches Programm – schnellste Lösung besteht darin, nur die betreffende Programmzeile als ASCII-Sequenz auf den Datenträger zu schreiben. Beim Laden des Programms erkennt dann der Basic-Interpreter, daß nicht tokenisiert wurde, und kann dies nachholen.

Bei größeren Programmen stößt dieses Verfahren allerdings schnell an seine Grenzen, weshalb Sie das Programm direkt in seiner Token-Form auf die Diskette oder Festplatte bringen. Spielen Sie also selbst einmal Basic-Interpreter, und zerlegen Sie die Programmzeile in ihr Token-Format.

Das erste Zeichen muß der Wert FFhex sein, da der Interpreter damit jedes Programm einleitet. Die beiden folgenden Byte sollen der Zeiger auf die nächste Zeilennummer sein. Diesen Wert können Sie erst eintragen, wenn Sie die Länge der Programmzeile kennen; jetzt lassen Sie ihn also noch frei. Die nächsten zwei Byte sollen die Zeilennummer codieren: Sie wollen die Zeilennummer 10 erzeugen und schreiben also hier die Bytes 0Ahex und 00hex in die Datei.

Als nächstes wird aus der Token-Tabelle der Code für den Cls-Befehl herausgesucht. Er lautet C0hex und bildet das sechste Byte des erzeugten Programms. Das Zeilenende markieren Sie durch eine nachfolgende Null. Bislang sind also sieben Byte für eine Programmzeile belegt. Da der Programmbeginn bei der relativen Adresse 0AF7hex liegt, muß der Zeiger auf die zweite Programmzeile also den Wert 0BFhex (0AF7hex + 7 = 0BFEhex) erhalten. An das Ende des Programms hängen Sie dann noch die obligatorischen Bytes 00hex, 00hex, 1Ahex und 00hex an, die den letzten Zeiger und das Ende des Programms markieren. Zusammengefaßt läßt sich das Programm also durch die Zahlenfolge

FF FE 0A 0A 00 C0 00 00 00 1A 00

darstellen. Das Listing »makecls.bas« sorgt dafür, daß diese Zeichen unter dem Namen »cls.bas« auf dem Datenträger gespeichert werden. Laden Sie die Datei einmal in den Speicher, und betrachten Sie das Listing. Sie werden sehen, daß das Verfahren funktioniert!

(Martin Althaus/sw)

```
[6U0] 10 CLS
[GD4] 20 INPUT "Dateiname? "; DNAME$
[WK3] 30 OPEN DNAME$ AS #1 LEN=1
[813] 40 FIELD #1,1 AS BYTE$
[6F1] 50 I%=0:J%=1
[4B2] 60 LAENGE%=LOF(1)
[KU0] 70 CLS
[L22] 80 WHILE I%<=LAENGE%
[C01] 90 GET#1
[LK3] 100 LOCATE J%,((I% MOD 16)*3+1)
[376] 110 PRINT RIGHT$("0"+HEX$(ASC(BYTE$)),2);"
";
[K14] 120 LOCATE J%,60+(I% MOD 16+1)
[AH7] 130 IF BYTE$<" " THEN PRINT ". "; ELSE PRINT
BYTE$;
[YG4] 140 IF INKEY$=" " THEN GOSUB 260
[ED1] 150 I%=I%+1
[7A4] 160 IF (I% MOD 16)=0 THEN J%=J%+1
[5S2] 170 IF J%<22 THEN 230
[C62] 180 LOCATE 23,25
[TW5] 190 PRINT "Bitte Leertaste drücken";
[KT1] 200 GOSUB 260
[D41] 210 CLS
[201] 220 J%=1
```

```
[P61] 230 WEND
[7M1] 240 CLOSE 1
[731] 250 END
[323] 260 WHILE INKEY$<>" "
[271] 270 WEND
[KM1] 280 RETURN
```

Listing 1. Mit dem Programm »seetoken.bas« können Sie Basic-Programme in ihrer Token-Schreibweise analysieren

```
[R24] 10 OPEN "cls.bas" AS #1 LEN=1
[NZ2] 20 FIELD #1,1 AS BYTE$
[652] 30 FOR I%=1 TO 11
[8X1] 40 READ WERT$
[755] 50 LSET BYTE$=CHR$(VAL("&h"+WERT$))
[F11] 60 PUT#1
[FE1] 70 NEXT I%
[CD1] 80 CLOSE 1
[N15] 90 DATA ff,fe,0a,0a,00,c0,00,00,00,1a,00
[P21] 100 END
```

Listing 2. »makecls.bas« ist ein Beispielprogramm für die automatische Generierung von Programmen

Beschleunigung von Basic-Programmen

Es gibt eine Menge Wege, Basic-Programme schneller zu machen. Die meisten ergeben sich direkt aus der Arbeitsweise des Basic-Interpreters. Vor allem im Bereich der Variablenverwaltung und der Programmstruktur lassen sich so beträchtliche Geschwindigkeitsgewinne erzielen.

Die folgenden Tips sind nur auf einen Geschwindigkeitsgewinn ausgerichtet und stehen einem sauberen Programmierstil eher im Wege. Aus diesem Grund sollten Sie die Tips nur bei geschwindigkeitskritischen Problemen einsetzen.

Einen ersten Ansatzpunkt zur Beschleunigung bietet die Programmstruktur: Stellen Sie daher Unterprogramme an den Anfang Ihres Programms.

Da der Basic-Interpreter beim Aufruf eines Unterprogramms mit dem Gosub-Befehl das Programm so lange durchsucht, bis er die nachfolgende Zeilennummer findet, sollte man Unterprogramme (vor allem, wenn sie häufig aufgerufen werden) immer an den Anfang eines Programms stellen. Mit einem Goto-Kommando auf den eigentlichen Programmbeginn lassen sich diese Passagen überbrücken. Statt

```
10 'Programmbeginn
...
5000 gosub 12000
...
11999 'Programmende
12000 'Unterprogramm
schreiben Sie also besser:
10 goto 1000
20 'Unterprogramm
...
999 return
1000 'Programmbeginn
...
5000 gosub 20
...
11000 'Programmende
```

Der nächste Trick zur Programmbeschleunigung besteht darin, die Kapazität von Programmzeilen auszunutzen, das heißt, sie vollzupacken. Die Verarbeitung jeder Zeilennummer kostet den Interpreter Zeit, da er die Zeiger auf jede Zeilennummer verarbeiten muß. Es ist schneller, möglichst viele Befehle in einer Zeile unterzubringen. Anstelle von

```
10 cls
20 print "Alles in Ordnung"
30 input na$
40 open na$ as #1 len 1
```

sollten Sie folgender Zeile den Vorzug geben:

```
10 cls:print"Alles in Ordnung":input na$:open na$ as #1 len 1
```

Auch Kommentare werden vom Basic-Interpreter im Programmtext codiert. Das heißt also, daß jeder Kommentar vom Interpreter als solcher erkannt und übergangen werden muß. Natürlich kostet dieser Vorgang Zeit. Daher entfernen Sie alle Kommentare aus dem (getesteten) Programm.

Auch die Verarbeitung jedes Leerzeichens kostet den Interpreter Zeit. Daher ist es sinnvoll, Leerzeichen an allen Stellen, an denen auf sie verzichtet werden kann, aus dem Programmtext herausnehmen. Die Zeile

```
100 if a=0 then a=50+s
```

ist zwar einfacher zu lesen, schneller ist folgende Zeile:

```
100 ifa=0thena=50+s
```

Bei jeder auftretenden Verzweigung prüft der Interpreter zunächst die Bedingung. Wenn sie erfüllt ist, wird zusätzlich das Sprungziel ausgewertet und ausgeführt. Diese Auswertung kostet Zeit. Strukturieren Sie Ihre Programme daher so, daß häufige Sprünge vermieden werden. Bei einer If-Abfrage soll das unwahrscheinlichere Sprungziel hinter dem »then« stehen und die wahrscheinlichere Möglichkeit im normalen Programm verarbeitet werden.

```
10 a%=rnd*10
20 if a%<=8 then 50
30 print "Die Zahl ist größer als 8"
40 stop
50 print "Die Zahl ist kleiner als 9"
```

Ein besseres Laufzeitverhalten zeigt die zweite Variante:

```
10 a%=rnd*10
20 if a%>8 then 50
30 print "Die Zahl ist kleiner als 9"
40 stop
50 print "Die Zahl ist größer als 8"
```

Basic kennt zwei Arten von Schleifen: Die For-Next- und die While-Wend-Schleife. Setzen Sie beide Schleifen richtig ein. Da die For-Next-Konstruktion schneller arbeitet als While-Wend, ist es günstiger, bei allen Anwendungen, bei denen die Abbruchbedingung von vornherein feststeht und nicht ständig im Programm abgefragt wird, auf die For-Next-Schleife zurückzugreifen.

```
10 open "date1.dat" for input as #1
20 while not eof(1)
...
1000 wend
```

Dieses Beispiel zeigt einen Weg, den Inhalt einer Datei zu lesen. Als schneller erweist sich jedoch die folgende Programmversion:

```
10 open "date1.dat" for input as #1
20 da=lof(1)
30 for za=1 to da
...
1000 next
```

Definieren Sie oft benötigte Variablen zuerst. Der Interpreter hält alle Variablen in einer Liste fest. Die Variablen, die zu Beginn definiert wurden, führen auch die Liste an. Da die Liste bei jedem Variablenzugriff durchsucht werden muß, bis der Name der Variablen gefunden ist, definieren Sie oft benötigte Variablen zu Beginn eines Programms. Dadurch sind sie in der Liste schnell zu finden. Der Interpreter verarbeitet zunächst nur die beiden ersten Zeichen eines Variablennamens. Nur wenn diese beiden Zeichen bei mehreren Variablen identisch sind, werden die anderen Zeichen zur Unterscheidung herangezogen. Diese zusätzliche Unterscheidung kostet Zeit. Wählen Sie daher die Variablennamen so, daß sie sich schon in den beiden ersten Zeichen unterscheiden, und beschränken Sie ihre Länge auf zwei Zeichen. Schreiben Sie die Programmzeilen

```
100 vorname$="Martin"
110 nachname$="Klappert"
120 fliesskomma#=1.234756
```

besser in folgender Form:

```
100 vo$="Martin":na$="Klappert":fl#=1.234756
```

Vermeiden Sie in Ihren Programmen Konstanten. Wenn Sie in einem Programm häufig Konstanten verwenden, so muß der Interpreter diese beim Ablauf des Programms auswerten und in sein internes Format umrechnen. Dies umgehen Sie dadurch, indem Sie anstelle der Konstante eine zu Beginn des Programms definierte Variable einsetzen. Statt

```
100 if name$="Erster Name im Text" then 5000120 if name$>"Erster Name im Text" then 6000
```

schreiben Sie besser:

```
10 nn$="Erster Name im Text"100 if na$=nn$ then 5000120 if na$>nn$ then 6000
```


Verzichten Sie auf mehrdimensionale Felder; sie kosten den Interpreter zum Speichern oder Laden von Daten viel Rechenzeit, weil zur Lokalisierung der Daten je nach Dimension viele interne Berechnungen nötig sind. Weichen Sie daher auf mehrere eindimensionale Felder mit unterschiedlichen Namen aus. So mag eine Definition zwar einfacher sein

```
10 dim n$(80,4)
```

doch auf folgende Weise geht es schneller:

```
10 dim n1$(80),n2$(80),n3$(80),n4$(80)
```

Schnellere Programme erhalten Sie auch, wenn Sie mathematische Operationen vereinfachen. Die Multiplikation und vor allem die Division und die Potenzierung kosten den Interpreter Zeit. Überprüfen Sie deshalb, ob Sie nicht schneller zum Ziel kommen, wenn Sie weniger zeitkritische Operationen verwenden. Die Multiplikation läßt sich beispielsweise durch Mehrfachaddition, die Division durch Multiplikation mit Werten kleiner als Null und die Potenzierung durch Mehrfachmultiplikation vereinfachen.

a*4 wird zu a+a+a+a, a/4 wird zu a*0.25, a↑3 wird zu a*a*a

Sollen Kommazahlen auf ihren ganzzahligen Wert reduziert werden, so ist es nur dann sinnvoll, den Int-Befehl einzusetzen, wenn der neue Wert einer Fließkommavariablen zugewiesen wird. Ansonsten runden Sie einfach ab, indem Sie die Kommazahl (Fließkommavariablen) einer Ganzzahl (Integervariablen) zuweisen. Die herkömmliche Methode sieht wie folgt aus:

```
10 a=1365.7866
20 b=int(a)
30 print b
```

Schneller wird diese Variante bearbeitet:

```
10 a=1365.7866
20 b%=a
30 print b%
```

Um schnelle Basic-Programme zu erhalten, ist es wichtig, die Variablentypen richtig einzusetzen. Basic erlaubt den Einsatz unterschiedlicher numerischer Variablentypen, wie Ganzzahlen (Kennung »%«), einfachgenaue Fließkommazahlen (Kennung »!«) und doppelgenaue Fließkommazahlen (Kennung »#«). Wird kein Variablentyp explizit angegeben, so verwendet Basic den Typ der einfachgenauen Fließkommazahl. Im Gegensatz zu einer Ganzzahl belegt dieser Datentyp vier anstelle von zwei Byte. Da das natürlich die Verarbeitungsgeschwindigkeit (besonders bei Vergleichen und bei arithmetischen Operationen) gegenüber der Verwendung von Ganzzahlen herabsetzt, sollte man immer prüfen, ob nicht eine Ganzzahl für eine Anwendung ausreicht. Wenn Sie in einem Programm nur Ganzzahlen verwenden, dann teilen Sie dies zu Beginn eines Programms dem Interpreter mit dem Befehl »defint a-z« mit.

Der Einsatz spezieller Befehlsformen entlastet den Basic-Interpreter

Bauen Sie einen If-Then-Befehl so auf, daß die am häufigsten vorkommende Bedingung am Anfang des Kommandos steht. Bei verknüpften If-Befehlen der Form

```
if a=0 and b=0 then
```

schreibt man besser

```
if a=0 then if b=0 then
```

weil der Interpreter dann unter Umständen (nämlich immer dann, wenn bereits die erste Bedingung nicht erfüllt ist) den zweiten Ausdruck nicht mehr auswerten muß. Werden beide Ausdrücke hingegen mit »and« verknüpft, so sieht sie der Interpreter als einen Ausdruck an und wertet beide aus.

Wird ein Next-Befehl mit nachfolgendem Variablennamen geschrieben, so überprüft der Interpreter, ob die genannte

Variable auch dem Variablennamen in der For-Schleife entspricht. Da diese Prüfung natürlich Rechenzeit benötigt, sollten Sie Variablennamen hinter dem »next« generell weglassen. Bei dieser Form schließt der Interpreter automatisch die letzte Schleife.

Der Bildschirmzugriff von Basic läuft gewöhnlich den »Umweg« über das BIOS (Basic Input Output System) des PC, um den Bildschirm zu adressieren und Zeichen darzustellen. Bei vielen Grafikkarten läßt sich der Bildschirmspeicher aber auch direkt adressieren. Dazu setzen Sie mit der Def-Seg-Anweisung das Datensegment auf den Beginn des Bildschirmspeichers und können dann mit »poke« und »peek« direkt in diesen schreiben oder aus ihm lesen.

Die Startadresse des Bildschirmspeichers ist bei der Hercules-Grafikkarte »&hb000« und bei der CGA- und EGA-Karte »&hb800«. Allerdings funktioniert dieses Verfahren nur bei solchen Grafikkarten, die über »Dual-Ported-RAM« verfügen und daher den Bildschirmzugriff nicht mit dem Zyklus des Kathodenstrahls synchronisieren müssen. Bei anderen Karten kann »Schnee« auf dem Bildschirm die Folge des direkten Speicherzugriffs sein.

Zum Schluß noch zwei Programme als Beispiel: Sie kopieren eine Datei in eine andere und entfernen dabei alle Steuerzeichen in der neuen Datei. Alle Zeichen, die in die neue Datei gelangen, werden auf dem Bildschirm angezeigt. In der ersten Version (Listing 1) ist das Programm nicht optimiert. In der zweiten Version (Listing 2) hingegen ist das Programm nach den genannten Gesichtspunkten optimiert und arbeitet daher wesentlich schneller.

(Martin Althaus/zi)

```
[XA5] 1 REM Programmname: DEMOLAHM.BAS
[HK4] 2 REM (C) 1989 Everts & Hagedorn
[ZD4] 3 REM DOS International
[B97] 10 REM Kopieren einer Datei in eine andere als
[WK7] 20 REM Demonstration der Befehle zur Dateiver-
[NP4] 30 REM waltung von PC-/GW-Basic
[RX7] 40 REM Entfernt beim Kopieren alle Steuerzeichen
[EU0] 50 CLS
[ST1] 60 ZAEHLER=1
[H57] 70 INPUT "Dateiname der Quelldatei ";QUELLES$
[DM6] 80 INPUT "Dateiname der Zieldatei ";ZIELS$
[SA3] 90 VIEW PRINT 10 TO 20
[F74] 100 OPEN QUELLES$ AS #1 LEN=1
[624] 110 FIELD#1,1 AS QUELLBYTES$
[RN4] 120 OPEN ZIELS$ FOR OUTPUT AS #2
[3S3] 130 WHILE ZAEHLER<LOF(1)
[G81] 140 GET#1
[FD3] 150 ZAEHLER=ZAEHLER+1
[BZ4] 160 IF ASC(QUELLBYTES$)<32 THEN 190
[SN3] 170 PRINT #2,QUELLBYTES$;
[FE3] 180 PRINT QUELLBYTES$;
[671] 190 WEND
[RA1] 200 CLOSE
[Z92] 210 VIEW PRINT
[Y21] 220 END
```

Listing 1. »demolahm.bas« ist zwar übersichtlich, doch nicht besonders schnell

```
[PT1] 1 CLS:L$=" ":INPUT "Dateiname der Quelldatei ";QU
$:INPUT "Dateiname der Zieldatei ";ZIS$:VIEW PRIN
T 10 TO 20:OPEN QU$ AS#1 LEN=1:DA=LOF(1):FIELD#1,
1 AS QB$:OPEN ZIS$ FOR OUTPUT AS #2:FOR ZA=1 TO DA
:GET#1:IF QB$<L$ THEN 3
[4Y3] 2 PRINT #2,QB$;:PRINT QB$;
[V34] 3 NEXT:CLOSE:VIEW PRINT
```

Listing 2. »demosprt.bas« ist die zeitoptimierte Version von Listing 1

Tips und Tricks

Programmieren Sie gerne in GW-Basic? Auf den nächsten Seiten finden Sie eine Fülle an Programmiertricks und Kniffen für Ihre eigenen Programme. Von anwenderfreundlichen Menüs über schnelle Sortierverfahren bis hin zu mathematischen Spielereien mit der Kreiszahl Pi ist für jeden Bedarf der eine oder andere Trick vorhanden. Insgesamt vierzehn Beispielprogramme runden die Tips- und Tricks-Seiten ab.

Als wenig ansprechend und kaum benutzerfreundlich müssen Programme bezeichnet werden, bei denen willkürlich nummerierte oder mit Buchstaben versehene Menüpunkte durch Drücken einer Zahl oder einer anderen »sinnlosen« Taste aufzurufen sind: »Bitte wählen Sie eine Funktion (Taste [1] bis [9]), oder drücken Sie [0], um das Programm zu beenden«. Dennoch findet man auch heute noch Software, bei der die Menüführung auf diese oder eine ähnlich unkomfortable Weise gelöst wurde. Dabei ist es wirklich einfach, Programme zu entwickeln, deren Menüpunkte sich durch eigene »Hotkeys« (in der Regel der Anfangsbuchstabe oder ein anderer leicht einzuprägender Buchstabe) aufrufen lassen, wie es zum Beispiel bei der Anwendungssoftware Microsoft Word oder Lotus 1-2-3 vorbildlich realisiert wurde.

Hotkey-Technik verleiht Ihren Programmen eine professionelle Menüführung

Um eine Taste im Programmablauf auf Gültigkeit zu prüfen und einem Unterprogramm zuzuordnen, werden alle Buchstaben und sonstige Tasten, die später als Hotkey oder Sondertasten zulässig sind, zuvor in einer String-Variablen zusammengefaßt (bei Verwendung von indizierten Menü-Variablenfeldern in der Reihenfolge Ihres Auftretens im Menü). Einzige Schwierigkeit: Gelegentlich ist es nicht einfach, einen geeigneten Namen für einen Menüpunkt zu finden, der einen bisher bereits verwendeten Anfangsbuchstaben besitzt. Dies ist jedoch notwendig, da Hotkeys eindeutige Tasten sein müssen.

Die Zusammenfassung aller gültigen Tasten in einer Zeichenkette hat den weiteren Vorteil, daß man sich später eine Vielzahl von If-Then-Abfragen spart. Angenommen, Sie haben für einige Buchstaben- und Zahlentasten Funktionen vorgesehen und wollen auch auf Drücken der Cursor-Tasten reagieren, so erfordert dies normalerweise eine Reihe von Vergleichen.

Nicht so bei der beschriebenen Methode: Mit Hilfe der Instr-Funktion kann ein Programm leicht prüfen, ob die gerade eingegebene Taste ein gültiger Hotkey ist (»instr <> 0«) und welche Unterfunktion mit dieser Taste in Verbindung steht. »instr« liefert Werte von 1 bis zu »len(String)« beziehungsweise 0, wenn die Taste ungültig ist. In nur zwei Zeilen wird abhängig von der gedrückten und gültigen Taste ein eigenes Unterprogramm aufgerufen:

```
a = instr(hotkey$, taste$): if a=0 then ...
on a goto ...
```

Vorsicht ist jedoch geboten, wenn Sondertasten mit erweitertem Tastaturcode integriert werden sollen, denn diese Tasten liefern einen Zwei-Zeichen-Code (»chr\$(0) + chr\$(n)«), der auch in den Hotkey-String integriert werden muß. Somit erfordern Sondertasten auch zwei Goto-Zeilennummern: Eine reguläre und eine »Dummy-Zeile«, die nur deshalb notwendig ist, um nachfolgende Zeichen im Hotkey-String richtig zuzuordnen.

Im Beispielprogramm »Leuchtbalken-Menü« (Listing 1) sind die Sondertasten [Cursor links] und [Cursor rechts] in den Hotkey-String integriert worden. Beachten Sie bitte in der On-Goto-Zeile die Sprungziele mit der nicht vorhandenen Zeilennummer 0. Dies sind Dummy-Einträge für die beiden Sondertasten. Das tatsächliche Sprungziel befindet sich jeweils direkt davor.

Das Demoprogramm enthält weiterhin eine kleine, bei Menüprogrammen häufig nützliche Unteroutine zum Zentrieren einer Zeichenkette innerhalb eines anderen Leerstrings. Zu übergeben ist die Länge des Zielstrings (Variable »center«) sowie der zu zentrierenden Zeichenkette in der Variablen »text\$«.

```
[MT1] 1 'MENU.BAS
[3G4] 1000 '*****
[AC7] 1010 '* Beispiel für HOTKEY-Programmierung*
[FN3] 1020 '*
[646] 1030 '* gültige Tasten werden den
[257] 1040 '* jeweiligen Menüpunkten zugeordnet *
[VN3] 1050 '*
[2F5] 1060 '* <ESC> beendet das Demo
[2G4] 1070 '*****
[PF9] 1080 RECHTS$=CHR$(0)+CHR$(77) : REM Scancode [Cur
sor rechts]
[639] 1090 LINKS$=CHR$(0)+CHR$(75) : REM Scancode [Curs
or links]
[K37] 1100 CENTER=9 : REM Länge der Menü-Strings
[9Y6] 1110 TEXT$="Files" : GOSUB 1600 : MENU$(1)=TEXT$
[8T6] 1120 TEXT$="Edit" : GOSUB 1600 : MENU$(2)=TEXT$
[6R6] 1130 TEXT$="Run" : GOSUB 1600 : MENU$(3)=TEXT$
[EG7] 1140 TEXT$="Options" : GOSUB 1600 : MENU$(4)=TEXT$
[UP9] 1150 HOTKEY$="fero" : REM Anfangsbuchstaben der M
enüpunkte
[8T9] 1160 HOTKEY$=HOTKEY$+" "+RECHTS$ : REM weitere gü
ltige Tasten
[ZW6] 1170 HOTKEY$=HOTKEY$+LINKS$+CHR$(13)+CHR$(27)
[8S2] 1180 CLS : COLOR 2,0
[MY2] 1190 LAST=1 : MENU=1
[8E2] 1200 FOR I=1 TO 4
[K05] 1210 PRINT TAB((I-1)*20+1) MENU$(I);
[E91] 1220 NEXT
[0X8] 1230 LOCATE 1,((MENU-1)*20+1) : COLOR 0,15 : PRIN
T MENU$(MENU)
[1V6] 1240 REM Schleife bis [Esc] gedrückt wird
[CS4] 1250 AS=INKEY$: IF AS="" THEN 1250
[3D7] 1260 MENU=INSTR(HOTKEY$,AS) : IF MENU=0 THEN 1250
[CF8] 1270 ON MENU GOTO 1280,1280,1280,1280,1310,1310,0
,1350,0,1390,1420
[3Z5] 1280 REM Menüpunkte 1 bis 4 (Hotkeys)
[NB5] 1290 IF MENU<>LAST THEN GOSUB 1450
[U22] 1300 GOTO 1490
[FA8] 1310 REM [Space] und Sondertaste [Cursor rechts]
[K76] 1320 MENU=LAST+1 : IF MENU>4 THEN MENU=1
[362] 1330 GOSUB 1450
[H22] 1340 GOTO 1520
[V76] 1350 REM Sondertaste [Cursor rechts]
[286] 1360 MENU=LAST-1 : IF MENU<1 THEN MENU=4
[L62] 1370 GOSUB 1450
[132] 1380 GOTO 1520
[NL2] 1390 REM [Enter]
[VA2] 1400 MENU=LAST
[X22] 1410 GOTO 1480
[272] 1420 REM [Esc]
[1U2] 1430 COLOR 3,0 : END
[MEB] 1440 REM Unterprogramm zur Hervorhebung des aktue
llen Menüpunktes
[128] 1450 LOCATE 1,((LAST-1)*20+1) : COLOR 3,0 : PRI
NT MENU$(LAST)
[0W8] 1460 LOCATE 1,((MENU-1)*20+1) : COLOR 0,15 : PR
INT MENU$(MENU)
[4U1] 1470 RETURN
[R7A] 1480 REM Unterprogramm zur Anzeige des aktuellen
Menüpunktes
[1D8] 1490 LOCATE 10,30 : COLOR 3,0 : PRINT "Menüpunk
t "MENU
[VA2] 1500 LAST=MENU
[E22] 1510 GOTO 1250
[RP6] 1520 REM Löschen der Menüpunkt-Anzeige
[FW6] 1530 LOCATE 10,30 : COLOR 3,0 : PRINT SPACE$(20)
[BB2] 1540 LAST=MENU
[Y22] 1550 GOTO 1250
[AH4] 1560 '*****
[Z47] 1570 '* Unterprogramm zur Zentrierung von *
[AR6] 1580 '* Strings innerhalb eines Strings *
```

Listing 1. »menu.bas« demonstriert Hotkey-Programmierung am Beispiel eines kleinen Leuchtbalkenmenüs


```
[PH4] 1590 '*****
[8A4] 1600 CENTERS$=SPACE$(CENTER)
[WE3] 1610 LAENGE=LEN(TEXT$)
[TT4] 1620 IF LAENGE>CENTER THEN 1650
[RD5] 1630 CENT=INT((CENTER-LAENGE)/2+.5)+1
[668] 1640 MID$(CENTERS$,CENT,LAENGE)=TEXT$ : TEXT$=CENTERS$
[YV1] 1650 RETURN
```

Listing 1. »menu.bas« demonstriert Hotkey-Programmierung am Beispiel eines kleinen Leuchtbalkenmenüs

Eine eigene Input-Routine gehört zu jedem anwendungsfreundlichen Basic-Programm

Die in GW-Basic implementierten Befehle und Funktionen zur Eingabe von Zeichen über die Tastatur sind alles andere als komfortabel und keinesfalls »narrensicher«. So ist es bei »input« beziehungsweise »line input« ohne weiteres möglich, den Cursor aus der vorgesehenen Zeile zu bewegen oder Texte einzugeben, die länger sind, als vom Programmierer vorgesehen. Fehlermeldungen wie »redo from start« zerstören unter Umständen die mühsam erzeugte Bildschirmmaske.

Das Input-Unterprogramm (Listing 2) bietet dagegen erhöhte Sicherheit (vor allem bei Maskenoperationen), wirkt optisch professionell und erlaubt alle notwendigen Editierfunktionen während der Eingabe. Obwohl es komplett in GW-Basic geschrieben ist und einen umständlichen, aber notwendigen Weg über »inkey\$« geht, ist dies kaum zu bemerken.

Das Programm erwartet als Eingabe eine Zeichenkette. Ziffern müssen daher vom Programmierer nachträglich mit der Val-Funktion in einen Zahlenwert umgewandelt werden.

Zur Verwendung der Routine: Positionieren Sie zunächst den Cursor mit »locate« auf die Stelle des Bildschirms, an der Sie die Eingabe wünschen, und stellen Sie mit der Variablen »maxlen« die maximale Länge der Zeichenkette ein, die angefordert oder editiert werden soll. Eine zu editierende Zeichenkette wird in der Variablen »in\$« übergeben und danach die Routine aufgerufen. Das Unterprogramm hebt daraufhin die zur Verfügung gestellte Eingabezeile inklusive der übergebenen Zeichenkette optisch hervor, so daß die maximale Länge der Eingabe sofort deutlich wird.

Der Anwender kann den übergebenen Text komfortabel bearbeiten. Zulässige Tasten zur Cursor-Bewegung sind [Cursor links], [Cursor rechts], [Home] und [End]. Zum Editieren stehen [Del] ([Entf]) und [Backspace] sowie die Tastenkombination [Ctrl-Y] zum Löschen des gesamten Feldes zur Verfügung. Weiterhin kennt das Unterprogramm einen Überschreib- und einen Einfügemodus, der mit [Ins] ([Einf]) umgeschaltet wird. Voreingestellt ist der Einfügemodus. Zur Rekonstruktion des übergebenen Textes während des Editierens genügt ein Druck auf die Escape-Taste.

Die Farben des Feldes während, vor oder nach der Bearbeitung lassen sich über die Variablen »locol1« und »locol2« beziehungsweise »hicol1« und »hicol2« (Vordergrund- und Hintergrundfarbe) zu Beginn des Programms voreinstellen.

Beim Editieren eines übergebenen Textes (»in\$ <> ""«) ist folgendes zu beachten: Ist die erste gedrückte Taste keine Cursor-Steuerungstaste, sondern zum Beispiel ein Buchstabe, so wird die (alte) Zeichenkette automatisch gelöscht, da das Programm von einer Neueingabe ausgeht. Dies nennt man auf »Computer-Chinesisch« auch: »first char clear« (»Löschen beim ersten Buchstaben oder Zei-

chen«). Wer diesen Effekt nicht wünscht, kann die dafür verantwortliche Zeile 1330 löschen.

Zuletzt sei noch darauf hingewiesen, daß das Demoprogramm durch ein leeres Feld »Namen« verlassen wird.

```
[MU1] 1 'INPUT.BAS
[2Z9] 10 CLS : LOCATE 4,25 : PRINT "Name:" : LOCATE 6,2
5 : PRINT "Vorname:"
[599] 20 LOCATE 8,25 : PRINT "Straße:" : LOCATE 10,25 :
PRINT "Ort:"
[AL9] 30 IN$=FELDS(1) : LOCATE 4,35 : MAXLEN=30: GOSUB
1000 : FELDS(1)=IN$
[BD5] 40 IF FELDS(1)=" THEN LOCATE 12,1: END
[UF9] 50 IN$=FELDS(2) : LOCATE 6,35 : MAXLEN=30 : GOSU
B 1000 : FELDS(2)=IN$
[BG9] 60 IN$=FELDS(3) : LOCATE 8,35 : MAXLEN=40 : GOSU
B 1000 : FELDS(3)=IN$
[NF9] 70 IN$=FELDS(4) : LOCATE 10,35 : MAXLEN=4 : GOSU
B 1000 : FELDS(4)=IN$
[CH9] 80 IN$=FELDS(5) : LOCATE 10,40 : MAXLEN=35 : GOSU
B 1000 : FELDS(5)=IN$
[ND1] 90 GOTO 30
[RM0] 1000 '
[ZE4] 1010 ' *****
[5A6] 1020 ' * KOMFORTABLE INPUT-ROUTINE: *
[CG6] 1030 ' * - KEIN WECHSEL DER ZEILE MÖGLICH *
[FC5] 1040 ' * - VORGEGBENE LÄNGEN *
[8A6] 1050 ' * - GUTE EDITIERMÖGLICHKEITEN *
[3U5] 1060 ' * - UNDO-FUNKTION MIT <ESC> *
[Y25] 1070 ' * - MASKEN-GEEIGNET *
[VF4] 1080 ' *****
[VN0] 1090 '
[V9A] 1100 IF LOCOL1=LOCOL2 THEN LOCOL1=3 : LOCOL2=0 :
HICOL1=0 : HICOL2=15
[6L6] 1110 TMP$=IN$ : INS=0 : FIRST=0 : SCANKEY=-1
[Y36] 1120 COL=POS(0) : LIN=CSRLIN : CPOS=COL
[5R3] 1130 COLOR HICOL1,HICOL2
[GA3] 1140 LOCATE LIN,COL,0
[7H6] 1150 PRINT TMP$;SPACES(MAXLEN-LEN(TMP$)+1);
[2N3] 1160 LOCATE LIN,CPOS,1
[F96] 1170 TASTE$=INKEY$ : IF TASTE$="" THEN 1170
[ZT6] 1180 TMPLN=LEN(TMP$) : TMPPTR=CPOS-COL+1
[EU9] 1190 IF LEN(TASTE$)=2 THEN SCANKEY=0 : FIRST=-1 :
ELSE SCANKEY=-1
[VU6] 1200 IF (SCANKEY=0) THEN GOSUB 1400 : GOTO 1380
[1N0] 1210 '
[7U6] 1220 REM [RETURN] [BACKSPACE] oder [ESC]?
[9N0] 1230 '
[R23] 1240 TMP=ASC(TASTE$)
[509] 1250 IF TMP=8 THEN GOSUB 1530 : GOTO 1380
: REM [BS]
[SSA] 1260 IF TMP=13 THEN IN$=TMP$ : GOSUB 1590 : RETURN
: REM [Return]
[UYA] 1270 IF TMP=27 THEN TMP$=IN$:CPOS=COL:FIRST=0:GOT
O 1380 : REM [Esc]
[SDA] 1280 IF TMP=25 THEN TMP$="":CPOS=COL:GOTO 1380
: REM [Ctrl-Y]
[1P0] 1290 '
[RV3] 1300 REM normale Zeichen
[4N0] 1310 '
[VF9] 1320 IF (TMP<32) THEN 1380 : REM keine Sonderzeic
hen zulassen
[5DB] 1330 IF (FIRST=0) THEN FIRST=-1 : TMP$="" : TMPLN
=0 : REM first char clear
[1T6] 1340 IF INS=0 THEN 1360 : REM Einfügemodus
[MYC] 1350 IF TMPPTR<=MAXLEN THEN MID$(TMP$,CPOS-COL+1,
1)=TASTE$ : CPOS=CPOS+1 : GOTO 1380
[CX4] 1360 IF TMPLN=MAXLEN THEN 1380
[C1D] 1370 TMP$=LEFT$(TMP$,CPOS-COL)+TASTE$+RIGHT$(TMP$
,TMPLN-(TMPPTR-1)) : CPOS=CPOS+1
[KF7] 1380 GOTO 1130 : REM ZURÜCK ZUR TASTEN-ABFRAGE
[4P0] 1390 '
[YF6] 1400 REM Auswertung für Sondertasten
[7N0] 1410 '
[6R4] 1420 SCAN=ASC(RIGHT$(TASTE$,1))
[XU9] 1430 IF SCAN=71 THEN CPOS=COL : RETURN
: [HOME]
[BBA] 1440 IF SCAN=79 THEN CPOS=COL+LEN(TMP$) : RETURN
: [END]
[WV9] 1450 IF SCAN=82 THEN INS=NOT INS : RETURN
: [INS]
[0NB] 1460 IF SCAN=75 THEN IF CPOS>COL THEN CPOS=CPOS-1
: RETURN : [left]
[6MC] 1470 IF SCAN=77 THEN IF (CPOS-COL)<TMPLN THEN CP
OS=CPOS+1 : RETURN : [right]
[6U8] 1480 IF SCAN>83 THEN RETURN
: [Del]
[AS7] 1490 IF (TMPLN=0) OR (TMPPTR>TMPLN) THEN RETURN
[759] 1500 TMP$=LEFT$(TMP$,TMPPTR-1)+RIGHT$(TMP$,TMPLN
-TMPPTR)
```

Listing 2. Die neue Input-Routine »input.bas« erlaubt professionelles Arbeiten mit Bildschirmmasken und vermindert Fehleingaben


```
[BV1] 1510 RETURN
[EN0] 1520 '
[D73] 1530 REM [BACKSPACE]
[PN0] 1540 '
[WA6] 1550 IF TMPPTR=1 THEN RETURN : REM POS(1)
[WF9] 1560 TMP$=LEFT$(TMP$,TMPPTR-2)+RIGHT$(TMP$,TMPLN-
-TMPPTR+1)
[2V3] 1570 CPOS=CPOS-1 : RETURN
[6P0] 1580 '
[0F6] 1590 REM Eingabezeile "normal" darstellen
[9N0] 1600 '
[5K6] 1610 COLOR LOCOL1,LOCOL2 : LOCATE LIN,COL,0
[156] 1620 PRINT IN$;SPACE$(MAXLEN-LEN(IN$)+1);
[PV1] 1630 RETURN
```

Listing 2. Die neue Input-Routine »input.bas« erlaubt professionelles Arbeiten mit Bildschirmmasken und vermindert Fehleingaben

So funktionieren selbstmodifizierende Programme mit der Tastaturpuffer-Technik

Jeder PC reserviert zur Pufferung der Eingaben über die Tastatur einen kleinen Speicherbereich, der maximal 16 Zeichen aufnehmen kann, und zwei 16-Bit-Speicherstellen, die dem Computer mitteilen, wo dieser Speicherbereich beginnt und wo er aufhört (im Segment 0000hex am Offset 041A/041Bhex und 041C/041Dhex). Durch Vergleich dieser Adressen stellt der Computer fest, ob und wieviele Zeichen im Tastaturpuffer gespeichert sind.

Diese Form der Speicherung kann sich jeder Programmierer zunutze machen und für eigene Zwecke verwenden. Hierzu werden per Poke-Befehl maximal 16 Zeichen in den Tastaturpuffer geschrieben und die Adressen für Beginn und Ende des Puffers manipuliert. Nach Beendigung des Programms oder einer Unterbrechung holt sich der Computer diese Zeichen aus dem Puffer und behandelt sie wie gewöhnliche Tastatureingaben. 16 Zeichen scheinen zur Programmierung sehr wenig, doch bei geschickter Anwendung ist dies mehr als genug.

Ein interessanter Einsatzbereich sind selbstmodifizierende Programme, Software also, die »sich selbst« verändert. Listing 3 und 4 liefern Beispiele dazu. Das erste Programm erzeugt (beziehungsweise verändert) eine Basic-Befehlszeile, die eine mathematische Funktion enthält (»def fn...«). Für diese dort abgelegte Funktion werden im Programm Werte berechnet.

Die Veränderung einer Basic-Zeile ist recht einfach. Kurz vor dem Programmende wird mit normalen Print-Befehlen eine gültige Basic-Befehlszeile mit Zeilennummer sowie eine Kommandozeile (Direktmodus) mit dem Befehl »run« auf dem Bildschirm ausgegeben. Dann wird der Cursor auf die Basic-Zeile positioniert und zweimal [Enter] in den Tastaturpuffer geschrieben. Damit werden beide Zeilen nach Programmende als »normale« Eingaben bestätigt und das Programm mit verändertem Listing erneut gestartet.

Anstelle des Befehls »run« könnte man natürlich auch »goto« verwenden, um das Programm ab einer bestimmten Zeilennummer neu zu starten.

Mit der beschriebenen Methode lassen sich maximal 16 Befehlszeilen bestätigen beziehungsweise ausführen.

Reicht dies für den geplanten Zweck nicht aus, helfen Programmschleifen weiter. Allerdings ist hierbei zu beachten, daß bei erneutem Start eines veränderten Programms (sowohl bei »run« als auch bei »goto«) der Inhalt aller bisher verwendeten Variablen verlorengeht, da Basic still und heimlich einen Clear-Befehl ausführt. Herkömmliche Schleifen wie »for i = 1 to n : ...« funktionieren also nicht. Es gibt zwei Methoden, dieses Problem zu umgehen:

- Die Zählervariable und andere für den Programmverlauf notwendige Daten werden in ungenutzte Speicher-

stellen abgelegt (»poke«), später wieder ausgelesen (»peek«), verändert, zurückgeschrieben und so weiter.

- Wichtige Variablen werden mit der Tastaturpuffer-Methode vor jedem Neustart im Direktmodus zugewiesen. Letzteren Weg beschreibt Listing 4, bei dem in einer Schleife RAM-Speicherbereiche ausgelesen und in Form von Data-Zeilen ins Programm eingebunden werden. Im Beispiel handelt es sich um den Speicherbereich der Bildschirmzeile 25 (Basic-Statuszeile). Natürlich hätte man Zeichencodes und Farbattribute auch mit der Screen-Funktion auslesen können, doch anhand dieses Beispiels ist leicht zu ersehen, daß sich mit der Tastaturpuffer-Technik prinzipiell beliebig große Speicherbereiche (aber natürlich auch andere Daten) in ein Programm einbinden lassen. Sinnvoll ist das Einbinden von Daten vor allen Dingen dann, wenn durch ihr Vorhandensein aufwendige Rechenoperationen bei späteren Programmläufen vermieden werden können.

Das Programm »buffer2.bas« (Listing 4) enthält auch eine kleine Routine zur automatischen Erkennung der aktiven Grafikkarte. Dazu wird die Speicherstelle 1040 (410hex) ausgelesen und die beiden Bits (4 und 5), die die Bildschirmkarte des Computers widerspiegeln, ausmaskiert. Die Funktion »(peek(1040) and 48)/16« liefert dann die Werte 0 bis 3 zurück, die folgende Grafikmodi repräsentieren:

- 0: keine Grafikkarte
- 1: 40 x 25 Zeichen Color
- 2: 80 x 25 Zeichen Color
- 3: 80 x 25 Zeichen Monochrom

Diese Speicherstelle beinhaltet darüber hinaus in den Bits 6 und 7 Informationen über die Anzahl der Diskettenlaufwerke (Minus 1) sowie die Größe des Arbeitsspeichers auf der Mutterplatine (Bits 2 und 3).

Weiter Hardwaredetails sind der benachbarten Speicherstelle 1041 zu entnehmen:

- Bit 0: nicht verwendet
- Bits 1 bis 3: Anzahl der seriellen Schnittstellen
- Bit 4: Joystickkarte angeschlossen
- Bit 5: nicht verwendet
- Bits 6 und 7: Anzahl der parallelen Schnittstellen

```
[512] 1 'BUFFER1.BAS
[4D8] 10 DEF FNA(X)=SIN(X)*12+COS(X)*13 : B$="sin(x)*12
+cos(x)*13"
[G21] 15 A$="fna(x) = "
[8U0] 20 CLS
[TS7] 30 LOCATE 5,25 : PRINT "Funktion: "A$;B$ : LOCATE
10,25
[2K6] 40 INPUT "Bitte Wert für x eingeben: ";X
[E88] 50 Y=FNA(X) : LOCATE 12,25 : PRINT "Lösung: Y = "
B$ = "Y"
[BMA] 60 LOCATE 15,20 : PRINT "Bitte geben Sie nun eine
neue Funktion ein."
[L4A] 70 LOCATE 17,22 : PRINT "[Enter] um das Programm
zu verlassen"
[52C] 80 LOCATE 5,25 : PRINT "Funktion: "SPACE$(30) : L
OCATE 5,35 : PRINT A$ : LOCATE 5,44
[9U1] 90 INPUT "",B$
[8T2] 100 IF B$="" THEN END
[3G4] 1000 '*****
[747] 1010 '*Beispiel für einen programmierten *
[CU4] 1020 '*Direktmodus *
[2T6] 1030 '* (Selbstmodifizierendes Programm) *
[XF6] 1040 '*Das Programm erzeugt eine neue *
[LD6] 1050 '*Zeile, in der eine mathematische *
[CU6] 1060 '*Funktion mit DEF FN definiert wird *
[2G4] 1070 '*****
[CT8] 1090 CLS : PRINT : PRINT "10 DEF "A$;B$ : B$="CH
R$(34)+B$+CHR$(34)
[S54] 1095 PRINT "RUN" : LOCATE 1,1
[8Z1] 2000 DEF SEG=0
[MH5] 2010 BUFLN=2 : REM 2 CRs notwendig
[LN5] 2020 POKE 1050,30 : POKE 1052,30+BUFLN*2
[PN3] 2030 FOR N=0 TO BUFLN-1
[V43] 2040 POKE 1053+(N*2)+1,13
[EP1] 2050 NEXT N
```

Listing 3. »buffer1.bas«, ein selbstmodifizierendes Programm, bei dem die zugrundeliegende mathematische Funktion fortlaufend verändert wird


```
[812] 1 'BUFFER2.BAS
[HM1] 10 GOTO 1000
[746] 900 DEF SEG=0 : KARTE=(PEEK(1040) AND 48)/16
[669] 905 IF KARTE=0 OR KARTE=3 THEN BLOCK=&HB0000 ELSE
BLOCK=&HB8000
[BX9] 910 DEF SEG=BLOCK : CLS : FOR I=0 TO 159 : READ A
: POKE I,A : NEXT
[LS2] 920 LOCATE 2,1 : END
[3G4] 1000 *****
[1U6] 1030 '* Selbstmodifizierendes Programm *
[BS6] 1040 '* Das Programm erzeugt DATA-Zeilen *
[4G6] 1050 '* mit dem Inhalt der 25.Bildschirm- *
[VS5] 1055 '* zeile und kopiert diese *
[GX6] 1060 '* schließlich in die erste Zeile *
[2G4] 1070 '*****
[W38] 1080 ZEILE=10 : SCHRITT=10 : START=24*160 : ENDE=
25*160
[VN0] 1090 '
[FV7] 1100 REM Einsprung für Tastaturpuffer-Schleife
[1M1] 1110 '
[P26] 1120 DEF SEG=0 : KARTE=(PEEK(1040) AND 48)/16
[GB9] 1130 IF KARTE=0 OR KARTE=3 THEN BLOCK=&HB0000 ELSE
BLOCK=&HB8000
[DV5] 1140 CLS : PRINT : PRINT ZEILE; "DATA";
[FU9] 1150 DEF SEG=BLOCK : REM automatische Grafikkarte
n-Erkennung
[B1B] 1160 FOR I= 0 TO 8 : PRINT PEEK(START+I); "," : NE
XT : PRINT PEEK(START+9)
[X57] 1170 START=START+10 : IF START>ENDE THEN RUN
[504] 1180 ZEILE=ZEILE+SCHRITT
[90B] 1190 PRINT "ZEILE="ZEILE":SCHRITT="SCHRITT":START
="START":ENDE="ENDE"
[SA6] 1200 PRINT : PRINT "GOTO 1120" : LOCATE 1,1
[G21] 1210 DEF SEG=0
[FP3] 1220 BUFLen=4 : REM 4 CRS
[VN5] 1230 POKE 1050,30 : POKE 1052,30+BUFLen*2
[VN3] 1240 FOR N=0 TO BUFLen-1
[353] 1250 POKE 1053+(N*2)+1,13
[PP1] 1260 NEXT N
```

Listing 4. »buffer2.bas«, Programmierung des Tastaturpuffers innerhalb von Schleifen mit frei bestimmtem Abbruchkriterium

Abfrage und Manipulation von Sondertasten in eigenen Programmen

Es gibt eine Reihe von Tasten, die – allein gedrückt oder in Kombination mit einer anderen Sondertaste – bei der Abfrage mit »inkey\$« keinen Wert zurückliefern. Dies sind die beiden Shift-Tasten, [Ctrl], [Alt], [Num Lock], [Scroll Lock] und [Caps Lock] sowie [Sys Req] bei einer AT-Tastatur. Es ist jedoch häufig wichtig zu wissen, ob und welche dieser Tasten gerade betätigt wird oder welchen Zustand die Lock-Tasten haben. Die Speicherstellen 1047 und 1048 im Segment 0 (0417/0418hex) geben darüber Auskunft.

Während »peek(1048)« die Betätigung der Lock-Sondertasten zeigt, können Sie mit Hilfe der Speicherstelle 1047 die Betätigung der Shift-Tasten, [Alt] und [Ctrl] sowie den Status der Lock-Tasten feststellen. Listing 5, eine Endlosschleife die sich nur mit [Ctrl-Break] beenden läßt, zeigt kontinuierlich den Zustand der einzelnen Bits, die jeweils mit einer Sondertaste in Verbindung stehen. Es hilft, die Abfrage dieser Tasten zu üben.

Der Zustand der Lock-Tasten läßt sich übrigens nicht nur ermitteln, er kann auch durch ein gezieltes »poke« in gewünschter Weise manipuliert werden. So können Sie beispielsweise durch

```
def seg = 0 : poke 1047, peek(1047) or 32
```

die Num-Lock-Taste aktivieren.

```
[BX1] 1 'SONDER.BAS
[NC3] 1000 *****
[EU4] 1010 '* Sondertasten-Abfrage *
[CS4] 1020 '* und Ermittlung ihres *
[BF4] 1030 '* aktuellen Status *
[5D3] 1040 '*****
[7U2] 1050 CLS : LOCATE 5,1
[LL9] 1060 PRINT "Peek(1047)" TAB(25) "Bit" TAB(40) "Pe
ek(1048)" TAB(65) "Bit"
[RK1] 1070 PRINT
[Z25] 1080 PRINT "[Shift rechts]" TAB(40) "----"
```

```
[HV5] 1090 PRINT "[Shift links]" TAB(40) "----"
[NZ5] 1100 PRINT "[Ctrl]" TAB(40) "[Sys Req]"
[HN6] 1110 PRINT "[Alt]" TAB(40) "[Ctrl-Num Lock]"
[BS7] 1120 PRINT "[Scroll Lock]" TAB(40) "[Scroll Lock]"
"
[MY6] 1130 PRINT "[Num Lock]" TAB(40) "[Num Lock]"
[B67] 1140 PRINT "[Caps Lock]" TAB(40) "[Caps Lock]"
[TY4] 1150 PRINT "[Ins]" TAB(40) "[Ins]"
[Y26] 1160 PRINT : PRINT "Wert:" TAB(40) "Wert:"
[502] 1170 DEF SEG=0
[DD4] 1180 A=PEEK(1047) : B=PEEK(1048)
[PD1] 1190 COL=6
[DE2] 1200 FOR I=0 TO 7
[K18] 1210 C=(A AND 2^I)/2^I : D=(B AND 2^I)/2^I : CO
L=COL+1
[R94] 1220 LOCATE COL,25 : PRINT C;
[EA4] 1230 LOCATE COL,65 : PRINT D;
[P91] 1240 NEXT
[926] 1250 LOCATE COL+2,24 : PRINT USING "###";A
[Q36] 1260 LOCATE COL+2,64 : PRINT USING "###";B
[WW1] 1270 GOTO 1180
```

Listing 5. »sonder.bas« zeigt, wie sich die Sondertasten in Basic abfragen lassen

Verschiedene Sortierverfahren in GW-Basic

Das Sortieren von Daten ist eine der wichtigsten Aufgaben der elektronischen Datenverarbeitung. Die bekanntesten Sortierverfahren lassen sich selbstverständlich auch in Basic realisieren. Bei der Umsetzung dieser Verfahren (Listings 6 bis 9) wurde darauf Wert gelegt, daß Start- und Endindex des zu sortierenden Variablenfeldes (»array\$«) für die Sortierung frei bestimmbar sind (Übergabe der Werte in den Variablen »astart« und »aende«). Auf diese Weise können auch Teilbereiche sortiert werden, was die Routinen noch universeller macht.

Bei den abgedruckten Listings handelt es sich um folgende Umsetzungen: Das klassische Bubble-Sort-Verfahren, eine optimierte Bubble-Sort-Version mit vorzeitigem Abbruchkriterium, Shell-Sort und Quick-Sort. Jedes dieser Verfahren hat seine besonderen Vorzüge und nach wie vor seine Daseinsberechtigung, denn nicht in allen Fällen wird Quick-Sort seinem Namen gerecht: Bei vorsortierten Feldern zeigt die optimierte Bubble-Sort-Routine ihre Stärken.

Quick-Sort ist die komplizierteste Methode, wie aus dem Vergleich mit den recht einfach strukturierten Listings der anderen Verfahren deutlich wird. Dafür benötigt sie nur wenig Vergleichsoperationen, was sich bei wachsenden Datenmengen positiv bemerkbar macht. Die Umsetzung des Quicksort-Algorithmus erfordert allerdings rekursive Programmierung – und das ist in Basic normalerweise nicht möglich. Zur Lösung des Problems arbeitet das Programm mit einem Pseudo-Stapelbereich (zwei dimensionierte, numerische Felder) zur Zwischenspeicherung wichtiger Zeiger. Er wird am Ende des Sortiervorgangs wieder gelöscht. Das bedeutet jedoch, daß sich mit Quicksort – zumindest in Basic – aus Speicherplatzgründen weniger Daten sortieren lassen als mit den anderen Verfahren.

Die Unterprogramme sind in der abgedruckten Fassung für den Einsatz eindimensionaler Felder vorgesehen, lassen sich jedoch leicht für mehrdimensionale Variablen anpassen. Das ist zum Beispiel notwendig, wenn Sie Adressen-Datensätze nach dem Inhalt eines Feldes sortieren wollen. In diesem Fall ändern Sie jeweils die Zeile mit dem Swap-Befehl in eine »Swap-Schleife«, zum Beispiel:

```
if array$(i,0) > array$(j,0) then for x=0 to 10 : swap
array$(i,x), array$(j,x) : next x
```

Zum Schluß noch ein Tip: Wenn der Speicherplatz zu knapp wird, um eine Sortierung im RAM durchzuführen, ist es auch möglich, die Sortierung sequentiell gespeicherter Daten aus der Datei »alt.dat« nach »sort.dat« auch mit dem Befehl

```
shell "sort <alt.dat >sort.dat"
```


von MS-DOS durchführen zu lassen oder besser noch von einem externen Quick-Sort-Programm, das den gesamten zur Verfügung stehenden Arbeitsspeicher des Computers nutzt.

```
[7V1] 1 'BUBBLE.BAS
[AL5] 10 DIM ARRAY$(10) : ASTART=0 : AENDE=10
[TN5] 20 FOR I=0 TO 10 : INPUT ARRAY$(I) : NEXT
[DS1] 30 GOSUB 1000
[VM5] 40 FOR I=0 TO 10 : PRINT ARRAY$(I) : NEXT
[7T0] 50 END
[3G4] 1000 *****
[EG6] 1010 ' * Unterprogramm BUBBLESORT *
[RL6] 1020 ' * Zu sortierendes Array: ARRAY$(I) *
[6A6] 1025 ' * Sortiert von ASTART bis AENDE *
[FG4] 1030 *****
[8G4] 1040 FOR I=ASTART TO (AENDE-1)
[VE3] 1050 FOR J=I+1 TO AENDE
[B19] 1060 IF ARRAY$(I)>ARRAY$(J) THEN SWAP ARRAY$(
I),ARRAY$(J)
[2S1] 1070 NEXT J
[4P1] 1080 NEXT I
[WV1] 1090 RETURN
```

Listing 6. Umsetzung des klassischen Bubble-Sort-Verfahrens in GW-Basic mit »bubble.bas«

```
[TZ1] 1 'BUBBLE2.BAS
[AL5] 10 DIM ARRAY$(10) : ASTART=0 : AENDE=10
[TN5] 20 FOR I=0 TO 10 : INPUT ARRAY$(I) : NEXT
[DS1] 30 GOSUB 1000
[VM5] 40 FOR I=0 TO 10 : PRINT ARRAY$(I) : NEXT
[7T0] 50 END
[3G4] 1000 *****
[GB6] 1010 ' * Unterprogramm BUBBLESORT V2 *
[RL6] 1020 ' * Zu sortierendes Array: ARRAY$(I) *
[5A6] 1025 ' * Sortiert von ASTART bis AENDE *
[LG4] 1040 *****
[B02] 1050 N=AENDE-1
[XT5] 1060 FOR I=N TO ASTART STEP-1 : PTR=0
[AF3] 1070 FOR J=ASTART TO N
[S06] 1080 IF ARRAY$(J)<ARRAY$(J+1) THEN 1110
[NZ4] 1090 SWAP ARRAY$(J),ARRAY$(J+1)
[FU1] 1100 PTR=J
[DR1] 1110 NEXT J
[1L5] 1120 N=PTR : IF PTR=0 THEN I=ASTART
[LN1] 1130 NEXT I
[BV1] 1140 RETURN
```

Listing 7. Das optimierte Bubble-Sort »bubble2.bas« eignet sich besonders für vorsortierte Felder

```
[5S1] 1 'SHELL.BAS
[AL5] 10 DIM ARRAY$(10) : ASTART=0 : AENDE=10
[TN5] 20 FOR I=0 TO 10 : INPUT ARRAY$(I) : NEXT
[DS1] 30 GOSUB 1000
[VM5] 40 FOR I=0 TO 10 : PRINT ARRAY$(I) : NEXT
[7T0] 50 END
[3G4] 1000 *****
[X46] 1010 ' * Unterprogramm SHELLSORT *
[4D6] 1020 ' * Zu sortierendes Array: ARRAY$(I) *
[TK6] 1025 ' * Sortiert von ASTART bis AENDE *
[FG4] 1030 *****
[FU3] 1040 DIST=AENDE-ASTART+1
[FL3] 1050 IF DIST<=1 THEN 1170
[953] 1060 DIST=INT(DIST/2)
[2L2] 1070 K=AENDE-DIST
[AM1] 1080 FLAG=0
[PD3] 1090 FOR I=ASTART TO K
[9C6] 1110 IF ARRAY$(I)<=ARRAY$(I+DIST) THEN 1140
[LB5] 1120 SWAP ARRAY$(I),ARRAY$(I+DIST)
[SR1] 1130 FLAG=-1
[RN1] 1140 NEXT I
[5B4] 1150 IF FLAG=-1 THEN GOTO 1080
[EW1] 1160 GOTO 1050
[RV1] 1170 RETURN
```

Listing 8. »shell.bas«: Shell-Sort ist ein weiterentwickeltes Sortierv erfahren

```
[RU1] 1 'QSORT.BAS
[WU5] 10 DIM ARRAY$(10) : ASTART=0 : AENDE=10
[TN5] 20 FOR I=0 TO 10 : INPUT ARRAY$(I) : NEXT
[DS1] 30 GOSUB 1000
```

```
[VM5] 40 FOR I=0 TO 10 : PRINT ARRAY$(I) : NEXT
[7T0] 50 END
[3G4] 1000 *****
[H36] 1010 ' * Unterprogramm QUICKSORT *
[4D6] 1020 ' * Zu sortierendes Array: ARRAY$(I) *
[SK6] 1030 ' * Sortiert von ASTART bis AENDE *
[LG4] 1040 *****
[DG9] 1050 DIM QSTART(INT((AENDE+1)/2+.5)),QENDE(INT((
AENDE+1)/2+.5))
[BX6] 1060 GOSUB 1070 : ERASE QSTART,QENDE : RETURN
[7W9] 1070 QSTART(ASTART)=ASTART : QENDE(ASTART)=AENDE
: K=ASTART-1
[CH6] 1080 K=K+1 : IF QSTART(K)>QENDE(K) THEN 1210
[S97] 1090 I=QSTART(K) : J=QENDE(K) : IF J<=I+1 THEN 12
10
[L26] 1100 MID=INT((I+J)/2) : TEMP$=ARRAY$(MID)
[222] 1110 IF I>J THEN 1190
[6M6] 1120 IF ARRAY$(I)<TEMP$ THEN I=I+1 : GOTO 1120
[VM6] 1130 IF ARRAY$(J)>TEMP$ THEN J=J-1 : GOTO 1130
[EZ2] 1140 IF I>J THEN 1190
[AB4] 1150 SWAP ARRAY$(I),ARRAY$(J)
[5W3] 1160 I=I+1 : J=J-1 : GOTO 1110
[ZS8] 1170 IF ARRAY$(I)<=ARRAY$(J) THEN SWAP ARRAY$(I),
ARRAY$(J)
[GW1] 1180 GOTO 1210
[918] 1190 QSTART(K+1)=QSTART(K) : QENDE(K+1)=J : GOSUB
1080
[PB9] 1200 QSTART(K+1)=QENDE(K+1)+1 : QENDE(K+1)=QENDE(
K) : GOSUB 1080
[9V2] 1210 K=K-1 : RETURN
```

Listing 9. Das in den meisten Fällen schnellste Sortierv erfahren: Quick-Sort in »qsort.bas«

Verzeichnisse unter GW-Basic lesen, sortieren und anzeigen

GW-Basic beinhaltet das Kommando »files «<spec>« zur Anzeige des Inhaltsverzeichnisses eines beliebigen Laufwerks. Leider hat der Programmierer jedoch keinen Einfluß auf die Ausgabe der Dateinamen, beispielsweise bei der Positionierung auf dem Bildschirm, bei der Bestimmung der Anzahl der Dateien pro Zeile und Spalte, bei der Reihenfolge und ähnlichem. Auch die Übernahme eines Dateinamens in eine Variable ist mit diesem Kommando nicht möglich. Daher muß eine eigene Unter routine zum Einlesen und Manipulieren einer Dateiliste geschrieben werden.

Listing 10 liest ein Verzeichnis in der von MS-DOS ge wohnen Art und Weise mit Angabe der Dateigröße, Da tum und Zeit der Dateierstellung und gibt dieses auf dem Bildschirm aus. Die Ausgabe kann auf Wunsch nach die sen Attributen sowie nach Namen oder Erweiterung sor tiert erfolgen. Der eigentliche Trick besteht darin, daß die MS-DOS-Befehle »dir« und »sort« über den Basic-Shell-Befehl verwendet werden und die Ausgabe dabei in eine Datei umgelenkt und zeilenweise gelesen und ange zeigt wird. Zugegeben, dies ist nicht besonders elegant, jedoch funktioniert es ohne viel Aufwand, denn was am Ende mit der erzeugten Verzeichnisseite passiert, darüber entscheidet der Programmierer allein. Neben der Darstel lung auf dem Bildschirm und der Übernahme bestimmter Inhalte für den weiteren Verlauf eines Programms läßt sich dieses Unterprogramm beispielsweise auch für eine einfache Diskettenverwaltung einsetzen.

```
[ZC1] 1 'DIR.BAS
[MD9] 10 SPEC$="*,*" : SORTFLAG=2 : REM Sortieren nach
Erweiterung
[RR2] 20 GOSUB 1000 : END
[3G4] 1000 *****
[RG6] 1001 ' * Unterprogramm zum Einlesen und *
[V56] 1002 ' * Anzeigen eines Directories *
[3S5] 1003 ' * Optional: Sortierung *
[7G4] 1004 *****
[ZET] 1005 FILE$="" : DATEI$="" : DIR.$$$" : SORT$="" |sor
t /+
[XX8] 1010 IF SORTFLAG=0 THEN SORT$="" : |kei
```



```

ne Sortierung
[N57] 1020 IF SORTFLAG=1 THEN SORT$=SORT$+"1" 'Name
[VN8] 1030 IF SORTFLAG=2 THEN SORT$=SORT$+"10" 'Erwe
iterung
[FW7] 1040 IF SORTFLAG=3 THEN SORT$=SORT$+"14" 'Größ
e
[7G7] 1050 IF SORTFLAG=4 THEN SORT$=SORT$+"24" 'Datu
m
[AM5] 1060 BEFEHL$="DIR "+SPEC$+SORT$+DATEI$
[PM2] 1070 SHELL BEFEHL$
[UZ4] 1080 OPEN "DIR.$$$" FOR INPUT AS #1
[VY2] 1090 WHILE NOT EOF(1)
[XK2] 1100 INPUT#1,DAT$
[CN7] 1110 IF INSTR(DAT$,"Datei(en)")<>0 THEN FILE$=D
AT$
[3Y8] 1120 IF MID$(DAT$,9,1)<>" " OR LEFT$(DAT$,12)="
DIR $$$ GOTO 1140
[S82] 1130 PRINT DAT$
[491] 1140 WEND
[6E2] 1150 PRINT FILE$
[125] 1160 CLOSE 1 : SHELL "erase dir.$$$"
[RV1] 1170 RETURN

```

Listing 10. »dir.bas«: Lesen und Anzeigen des Verzeichnisses
- auf Wunsch auch sortiert

Arbeiten mit dem binären Zahlensystem

GW-Basic verarbeitet Zahlen auf der Basis des Oktal-, Dezimal- und des Hexadezimalsystems. Häufig – besonders beim Interpretieren und Manipulieren von Speicherstellen, die bitweise codiert sind – wäre das Arbeiten mit dem binären Zahlensystem erheblich sinnvoller.

Die Programme »bin.bas« und »bin\$.bas« (Listings 11 und 12) sind kurze Unterprogramme, die eine Binärzeichenkette (zum Beispiel »1010111«) in dezimale Werte wandeln und umgekehrt. Bei der Wandlung binär-dezimal (»bin.bas«) wird die Zeichenkette in der Variablen »bin\$« übergeben, das dezimale Resultat steht nach Ausführung in der Variablen »bin«. Bei »bin\$.bas« verhält es sich genau umgekehrt: Der dezimale, numerische Wert in »bin« liefert den Zeichenkette »bin\$«.

Bei der Wandlung von hexadezimal nach dezimal und zurück kann man auf implementierte Funktionen in Basic zurückgreifen. So liefert »hex\$(«) eine Zeichenkette mit dem hexadezimalen Wert einer Dezimalzahl. Bei der Rückwandlung hilft die Funktion val():

```

a$ = hex$(4096)
dec = val("&h"+a$)

```

```

[HC1] 1 'BIN.BAS
[5L7] 10 BIN$="1111111111111111" : GOSUB 1000 : PRINT B
IN : END
[NH4] 990 *****
[G17] 991 ' Unterprogramm zur Umwandlung von *
[W46] 992 ' Binärzahlen in String BIN$ *
[8S4] 993 ' (maximal 16 Bit!) *
[9K4] 995 *****
[4B1] 1000 BIN=0
[W34] 1010 FOR I=0 TO LEN(BIN$)-1
[MW6] 1020 BIN=BIN+(MID$(BIN$,LEN(BIN$)-I,1))*2^(
I)
[C91] 1030 NEXT
[8V1] 1040 RETURN

```

Listing 11. Umwandlung von Binärzeichenketten in dezimale numerische Werte: »bin.bas«

```

[DL1] 1 'BIN$.BAS
[H56] 10 BIN=65535! : GOSUB 1000 : PRINT BIN$ : END
[NH4] 990 *****
[G17] 991 ' Unterprogramm zur Umwandlung von *
[VM6] 992 ' Integerzahlen in Binärzahlen *
[YA4] 993 ' (0 bis 65535) *
[9K4] 995 *****
[Z39] 1000 BIN$="" : HI=INT(BIN/256) : LO=BIN-HI*256 :
IF HI=0 THEN 1020
[4HB] 1010 FOR A=7 TO 0 STEP-1 : BIN$=BIN$+RIGHT$(STR$(
(HI AND 2^A) = 2^A),1) : NEXT
[YHB] 1020 FOR A=7 TO 0 STEP-1 : BIN$=BIN$+RIGHT$(STR$(
(LO AND 2^A) = 2^A),1) : NEXT
[H13] 1030 A=INSTR(BIN$,"1")
[DY8] 1040 IF A>0 THEN BIN$=RIGHT$(BIN$,LEN(BIN$)-A+1)

```

```

ELSE BIN$=""
[CV1] 1050 RETURN

```

Listing 12. Mit »bin\$.bas« lassen sich Dezimalzahlen binär (als String) darstellen

Mathematische Näherungslösungen in GW-Basic

Die drei Programme »ggt.bas«, »integral.bas« und »pi.bas« (Listings 13 bis 15) sind kurze Beispiele für Umsetzungen zahlreicher mathematischer Probleme in GW-Basic. Mit ihrer Hilfe lassen sich der größte gemeinsame Teiler zweier Zahlen, das Integral einer mathematischen Funktion und die Kreiszahl Pi durch Näherungs- beziehungsweise Intervallverfahren bestimmen. Nach gleichem oder zumindest sehr ähnlichem Prinzip werden auch Wurzeln oder Nullstellen einer Funktion berechnet. Alle Listings sind durch Hinweistexte dokumentiert. (Gerhard Torf/tr)

```

[4D1] 1 'GGT.BAS
[RS3] 10 ZAHL1=19612 : ZAHL2=-144
[BS1] 20 GOSUB 1000
[WZA] 30 PRINT "Größter gemeinsamer Teiler der Zahlen"Z
AHL1"und "ZAHL2;
[TL3] 40 PRINT "ist: "GGT : END
[UG4] 900 *****
[5Y6] 910 ' Unterprogramm zur Bestimmung des *
[W97] 920 ' größten gemeinsamen ganzzahligen *
[HK5] 930 ' Teilers zweier Werte. *
[CA5] 940 ' VAR: ZAHL1 und ZAHL2 *
[MX4] 950 ' Resultat: GGT *
[CH4] 960 *****
[Z28] 1000 Z1=ABS(ZAHL1) : Z2=ABS(ZAHL2) : IF Z1<Z2 THE
N SWAP Z1,Z2
[CX7] 1010 HILF = Z1-Z2 : IF Z2<HILF THEN Z1=HILF : GOT
O 1010
[WZ5] 1020 Z1=Z2 : Z2=HILF : IF Z1<>Z2 THEN 1010
[X33] 1030 GGT=Z1 : RETURN

```

Listing 13. Den größten gemeinsamen Teiler zweier numerischer Werte berechnet das Programm »ggt.bas«

```

[KF2] 1 'INTEGRAL.BAS
[RX3] 1000 *****
[9H5] 1010 ' Integralberechnung einer *
[7H5] 1020 ' mathematischen Funktion *
[D75] 1030 ' im Bereich INT1 bis INT2 *
[YC5] 1040 ' und der Anzahl der Inter- *
[3B4] 1050 ' valle: ANZAHL *
[FY3] 1060 *****
[A41] 1070 CLS
[3C3] 1080 DEF FNA(X)=2*X^3+4
[1V4] 1090 INT1=1 : INT2=3 : ANZAHL=2^10
[BH9] 1100 PRINT "Anzahl der Intervalle: "ANZAHL TAB(30)
) ", Fläche: ";
[172] 1110 FLAECHE#=0
[PX4] 1120 ABSTAND#=(INT2-INT1)/ANZAHL
[3V1] 1130 X1#=INT1
[K13] 1140 X2#=X1#+ABSTAND#
[BW5] 1150 TEIL#=ABSTAND#*(FNA(X1#)+FNA(X2#))/2
[974] 1160 FLAECHE#=FLAECHE#+TEIL#
[SL6] 1170 X1#=X2# : IF X1#+ABSTAND#<=INT2 THEN 1140
[1Y2] 1180 PRINT FLAECHE#

```

Listing 14. Die Integralrechnung durch Trapezberechnung im Intervallverfahren demonstriert »integral.bas«

```

[T51] 1 'PI.BAS
[8E4] 2 *****
[VZ6] 3 ' Programm zur Berechnung von Pi durch *
[RH6] 4 ' Annäherung eines 6-Ecks an einen *
[Y75] 5 ' Kreis (hier: 6*2^40 Ecken) *
[GE4] 6 *****
[7B8] 10 FOR I=0 TO 40 : REM abhängig von Rechengenauig
keit
[SH6] 20 HILF=ABS(W2#-0) : REM nur 1. Durchlauf = 1
[VB3] 30 W1#=2*SQR(3)*HILF+W1#
[0A2] 40 W2#=3*HILF+W2#
[ZD3] 50 W1#=2*W1#*W2#/(W1#+W2#)
[HD2] 60 W2#=SQR(W1#*W2#)
[FL9] 70 PRINT 6*(2^I);TAB(30) W2#;W1# : REM Näherungsw
erte ausgeben
[G51] 80 NEXT

```

Listing 15. »pi.bas« zeigt, wie sich die Kreiszahl Pi durch Annäherung eines Vielecks an einen Kreis immer genauer bestimmen läßt

SBasic 1.0

Ein Basic-Compiler zum Nulltarif

Beinahe zu jedem PC wird serienmäßig neben MS-DOS auch der Basic-Interpreter GW-Basic mitgeliefert. Doch die Geschwindigkeit dieses Basic-Dialektes ist für viele Anwendungen nicht ausreichend. Abhilfe schafft der Compiler »SBasic«, der einen Großteil des GW-Basic-Befehlssatzes beherrscht und die in Frage kommenden Programme in schnelle, direkt ausführbare Exe-Dateien übersetzt.

Professionelle Basic-Compiler sind nicht gerade billig und oft so hochgezüchtet, daß sie normale GW-Basic-Programme gar nicht mehr ohne größere Änderungen verarbeiten. Deshalb bieten wir Ihnen mit dem Basic-Compiler »SBasic« eine preiswerte Alternative, die einen Großteil des GW-Basic-Befehlssatzes beherrscht und diesen ohne Änderungen übersetzt. Die drei Befehlsbereiche die (noch) fehlen, sind alle Grafikanweisungen und die Kommandos für die Ereignis- und Fehlerbehandlung. In vielen Fällen können Sie jedoch auf diese drei Befehlsgruppen getrost verzichten.

SBasic eignet sich besonders gut zum Programmieren von kleinen Utilities, die bei herkömmlichen Compilern die ganze Laufzeitbibliothek angehängt bekommen und deshalb recht schnell viele KByte groß werden; bei SBasic ist dies nicht der Fall, so daß Sie viel Speicherplatz auf der Diskette oder Ihrer Festplatte sparen. Im Gegensatz zu anderen Basic-Compilern verarbeitet SBasic keine ASCII-Dateien (diese werden mit »save name«, »a« gespeichert), sondern nur die regulären GW-Basic-Dateien, die im Token-Format (auch als Binärmodus bezeichnet) gespeichert sind. Dieses Verfahren hat den Vorteil, daß einige wichtige Konvertierungsarbeiten schon vom GW-Basic-Interpreter geleistet werden und damit der Umfang des SBasic-Compilers geringgehalten werden kann.

Die nicht vorhandene Fehlerbehandlung von SBasic bringt allerdings eine weitere Einschränkung: SBasic ist kein Compiler zum Testen von Programmen, sondern ein »Nachbrenner« für bereits funktionsfähige Basic-Programme.

Doch nach den Schwächen von SBasic, die wir Ihnen aufgrund der vielen Vorzüge ganz offen eingestehen, nun zu den positiven Seiten. Zunächst ist hier der Geschwindigkeitszuwachs der Compile (das Ergebnis nach dem Compilieren eines Quelltextes) zu nennen, der sich durchaus im für Basic-Compiler üblichen Rahmen bewegt. Zudem haben Sie unter

SBasic mehr Speicherplatz zur Verfügung. Wie Sie vielleicht wissen, legt GW-Basic alle Daten (Systemvariablen, Programmtext, Variablen, Felder und Zeichenketten) im gleichen Segment ab. Dadurch stehen Ihnen maximal 64 KByte für die Programmierung zur Verfügung, die bei größeren Projekten sehr schnell belegt sind.

Der SBasic-Compiler kann im Gegensatz zu GW-Basic den gesamten Arbeitsspeicher Ihres PC nutzen

Weil SBasic jedoch mit mehreren Segmenten arbeitet (das ist bereits an der Exe-Endung der compilierten Programme erkennbar), können Sie den gesamten Arbeitsspeicher Ihres PC nutzen, und zwar nach folgendem Schema:

- 64 KByte für den Programmcode,
- 64 KByte für alle numerischen Variablen einschließlich der Felder,
- 1024 Byte für den Stack,
- den restlichen Speicher für Strings (Zeichenketten) und Stringarrays (Zeichenkettenfelder).

Die maximale Anzahl der Programmzeilen ist in der Variablen »maxzeil« in Zeile 112 und die maximale Anzahl der Variablen in der Variablen »maxvar« in Zeile 114 des

SBasic-Quelltextes festgelegt. Diese beiden Parameter können Sie nach Belieben an Ihre Bedürfnisse anpassen.

Die Strings werden von SBasic gänzlich anders verwaltet, als dies beim GW-Basic-Interpreter der Fall ist. Jeder String belegt konstant 256 Byte und beginnt immer an einer Segmentadresse. Das erste Byte enthält die Stringlänge. Dieses Verfahren geht zwar mit dem verfügbaren Speicherplatz sehr großzügig um, doch macht es zeitaufwendige dynamische Speichervergabe-Algorithmen und Garbage-Collection-Vorgänge überflüssig. Ein weiterer Vorteil von SBasic, den vor allem Assembler-Unkundige zu schätzen wissen werden, ist das Einbinden von Maschinencode-Unterprogrammen. Im Gegensatz zu anderen Compilern, bei denen Assembler-Programme erst an ein neues Format angepaßt werden müssen, verarbeitet SBasic die Maschinencode-Routinen im üblichen GW-Basic-Format.

Es gibt allerdings zwei Einschränkungen, die die Strings betreffen. Zum einen muß ein String, in dem ein Maschinencode-Unterprogramm abgelegt ist, oder der als Parameter übergeben wird, im Zugriffsbereich des Datensegments liegen. Ist dies nicht der Fall, so bekommt der Anwender den Fehler »String außer Reichweite« angezeigt. Definieren Sie in einem solchen Fall den String einfach früher im Programm, um ihn näher an das Datensegment zu legen. Das zweite Problem tritt bei Maschinencode-Routinen auf, die mit String-Feldern arbeiten. Weil SBasic keine Deskriptoren kennt, muß es diese simulieren, und das geht natürlich nur bei einzelnen Strings gut, bei String-Feldern funktioniert es nicht. Um den SBasic-Compiler zu erzeugen, müssen Sie zuerst den Quelltext in Listing 1 abtippen. Darauf haben Sie die Wahl. Entweder geben Sie den langen Assembler-Quelltext der SBasic-Laufzeitbibliothek (»Runtime

Hier übersetzt
sich SBasic selbst

```
SBasic 1.0, Compiler für GW-Basic, (C) 1989 Euerts&Hagedorn, DOS International
Quelldatei [L.BAS]: sbasic
Zeile: 1798 (13584)
Bibliothek: 5362
Programm: 36882
Stack: 1024
Variablen: 17858
Strings: 78848
Zeile: 18 (189)
```


Library») ein und assemblieren ihn mit dem MASM (Microsoft-Assembler) ab Version 4.0 oder TASM (Turbo-Assembler), oder Sie tippen das wesentlich kürzere Hexdump der assemblierten Laufzeitbibliothek aus Listing 3 ab und konvertieren es mit der Eingabehilfe »SCheck« (das erste Listing in dieser DOS-Extra-Ausgabe) in das erforderliche binäre Format. Das Abtippen von Listing 3 bedeutet zwar wesentlich weniger Arbeit, doch wenn Sie über Listing 2 verfügen, können Sie dafür die Laufzeitbibliothek problemlos modifizieren und erweitern.

Um Maschinencode-Unterschiede mit SBasic zu beheben, müssen Sie keine Anpassungen vornehmen

Mit den folgenden Befehlen übersetzen Sie Listing 2, das Sie mit »sbasic.asm« bezeichnen müssen, in das erforderliche Binärformat:

```
masm sbasic;
link sbasic;
exe2bin sbasic
```

Wenn Sie den Turbo-Assembler von Borland besitzen, ersetzen Sie »masm« und »link« durch »tasm« und »tlink«. Die Umgehung von »exe2bin« mit der Option »/t« von »tlink« funktioniert allerdings nicht, weil in »sbasic.asm« weder ein Einsprungspunkt angegeben ist, noch die Startadresse des Programms bei 100hex beginnt.

Nachdem Sie den SBasic-Quelltext »sbasic.bas« abgetippt und die Laufzeitbibliothek im Binärformat als »sbasic.bin« vorliegen haben, können Sie SBasic sich selbst compilieren lassen, um die Geschwindigkeit des Compilers zu erhöhen. Dazu muß die Datei »sbasic.bas« allerdings im Token-Format vorliegen, denn im ASCII-Format funktioniert die Compilierung nicht.

Doch womit können Sie SBasic compilieren? Ganz einfach – mit GW-Basic! Starten Sie dazu GW-Basic und SBasic mit

```
gwbasic sbasic
und geben Sie bei der Frage nach der zu compilierenden Datei
```

```
sbasic
ein. Darauf startet die Compilierung (Bild), die geraume Zeit in Anspruch nimmt – allerdings nur beim ersten Mal!
```

Auf einem herkömmlichen 10-MHz-AT dauert dieser Vorgang etwa 30 Minuten. Deshalb müssen Sie schon etwas Geduld mitbringen. Doch keine Sorge, Sie werden für das Warten reichlich entschädigt. Wenn Sie nämlich SBasic als ausführbare Exe-Datei vorliegen haben und das Programm nochmals mit

sich selbst compilieren (dies ist übrigens ein allgemein anerkannter guter Test für die Stabilität eines Compilers), dauert der Vorgang auf einem 10-MHz-AT nur noch eine Minute und 45 Sekunden. Die Compilierung erfolgt jetzt also mehr als 17mal so schnell. Ein eindrucksvoller Beweis für die Leistung von SBasic!

Wundern Sie sich übrigens nicht, wenn nach dem zweiten Compilieren das SBasic-Ergebnis ein Byte kürzer geworden ist. Der Grund ist das Dateiende-Zeichen mit dem ASCII-Wert 26, das der GW-Basic-Interpreter im Gegensatz zu der compilierten Version von SBasic an das Ur-Compilat angehängt hat.

Wenn Sie den SBasic-Quelltext von SBasic selbst compilieren lassen, arbeitet das Ergebnis über 17mal so schnell

Wenn Sie beim ersten Compilieren des SBasic-Quelltextes trotzdem nicht über eine halbe Stunde warten möchten, dann speichern Sie das Programm mit

```
save "sbasic",a
```

im ASCII-Format ab und lassen es von einem anderen Compiler (Quick Basic oder Turbo Basic) ausführen. Zuvor müssen Sie aber noch die Anweisung

```
eindat$=command$(1)
```

in Zeile 103 entfernen, weil sowohl Quick Basic als auch Turbo Basic den Sinn dieser Befehlszeile mißverstehen. Erst wenn Sie das SBasic-Compilat nochmals seinen Quelltext compilieren lassen, dürfen Sie die Anweisung wieder einfügen, damit Sie fortan beim Aufruf von SBasic in der Kommandozeile auch den Dateinamen des zu compilierenden Programms angeben können.

Während des Compiliervorganges zeigt SBasic immer die aktuell bearbeitete Zeile und in Klammern deren Zeilennummer im Quelltext an. Zur Halbzeit gibt SBasic eine kleine Statistik des zu compilierenden Programms aus. Diese Ausgabe erfolgt deswegen zur Halbzeit, weil es sich bei SBasic um einen Two-Pass-Compiler handelt. Gegenüber professionellen One-Pass-Compilern entfällt somit das Anlegen einer platzfressenden Vorwärtsreferenzliste; dafür arbeitet ein Two-Pass-Compiler allerdings auch langsamer.

Prioritätsebene	Operatoren
1	»-« (negatives Vorzeichen), »not«
2	»↑« (Potenz)
3	»*«, »/«, »\« (Integer-Division), »mod« (Rest einer Integer-Division)
4	»+«, »-« (Addition und Subtraktion)
5	»>«, »<«, »=«, »>=«, »<=«, »><« (Vergleichsoperatoren)
6	»and«, »or«, »xor« (logische Operatoren)

Tabelle 1. Die Operatoren von SBasic und ihre Prioritätsebenen

Gegenüber GW-Basic neu hinzugekommen ist bei SBasic die Funktion »command\$()«. »command\$« ist ein String-Feld, das Parameter enthält, die beim Programmaufruf in der Kommandozeile übergeben werden. Der Index beginnt ab 1, »command\$(0)« ist immer leer. Alle Kleinbuchstaben werden in Großbuchstaben umgewandelt, die Umlaute allerdings nicht. Zur Trennung der Parameter dient eine beliebige Anzahl von Leerzeichen, ein Komma oder der Schrägstrich. Letzterer erscheint zusätzlich im String.

Im folgenden ein Beispiel für den Aufruf eines Programms mit dem Namen »test«:

```
test first,, third fourth/fifth
```

Nach diesem Programmaufruf enthält die Variable »command\$(1)« den Text »first«, »command\$(2)« bleibt aufgrund des Doppelkommata leer, »command\$(3)« enthält den Text »third«, »command\$(4)« ist auf »fourth« gesetzt und »command\$(5)« enthält den Text »fifth«.

Daß die Command-Funktion wirklich funktioniert, können Sie sofort feststellen, indem Sie in der MS-DOS-Kommandoebene eingeben:

```
sbasic sbasic
```

Dadurch compiliert sich SBasic wieder selbst, allerdings werden Sie gar nicht mehr nach dem Namen der zu compilierenden Datei gefragt – den haben Sie nämlich bereits beim Aufruf angegeben.

SBasic enthält gegenüber GW-Basic eine zusätzliche Funktion, die die Übergabe von Parametern in der Kommandozeile erlaubt

Weil ein von SBasic compiliertes Programm mehr Speicher als die entsprechende Interpreter-Version adressieren kann, läßt sich die Kapazität von SBasic durch folgenden Trick erhöhen: Erzeugen Sie eine zweite Version, in der Sie die Maximal-Parameter in den Zeilen 112 bis 123 an Ihre persönlichen Bedürfnisse anpassen. So ist es zum Beispiel ratsam, die Anzahl der For-Next-Schleifen und der Data-Zeilen zu erhöhen.

Anweisung	Erklärung (und Besonderheiten gegenüber GW-Basic)	Anweisung	Erklärung (und Besonderheiten gegenüber GW-Basic)
Befehle:			open "name" for append as #nummer [len=satzlänge] Für »satzlänge« sind maximal 128 Byte zulässig; maximal drei Dateien können gleichzeitig geöffnet sein; speziell behandelte Geräte sind »kybd:«, »scrn:«, »com1:« und »com2:«; für die serielle Schnittstelle ist der Eingabepuffer immer 64 Byte groß, »DTR« und »RTS« sind immer aktiv, standardmäßig sind acht Datenbits, keine Parität und ein Stopbit eingestellt, nur die Baudrate läßt sich verändern, auch »krumme« Baudraten von 3 bis 65535 sind erlaubt)
BEEP	Ton ausgeben	OUT	Wert auf Port ausgeben
BLOAD	Speicherblock lesen	PLAY	Musik spielen (die Hintergrundmodusanweisung »MB« wird ignoriert; Pausen müssen keine Längenangaben haben)
BSAVE	Speicherblock schreiben	POKE	Wert in Speicher schreiben
CALL	Maschinenprogramm aufrufen Der Call-Befehl des SBasic-Compilers ist kompatibel zum Call-Befehl des GW-Basic-Interpreters. (Bei Stringparametern kann jedoch der Fehler »String außer Reichweite« auftreten. In diesem Fall sollten die Strings am Anfang des Programms verwendet (definiert) werden, damit SBasic sie näher an das Datensegment legt. Stringfelder sind als Parameter nicht erlaubt. Ebenso sind Längenveränderungen von Strings in Maschinen-code-Unterprogrammen nicht zulässig)	PRINT, LPRINT	Werte oder Text ausgeben (die Zusatzanweisungen »tab« und »spc« werden unterstützt, aber die Sonderform »print using« und Ausgabeumleitungen sind nicht erlaubt. Wenn Sie in SBasic mehrere Werte durch Kommata getrennt in eine Datei schreiben möchten, müssen Sie dazu anstelle von »print #« den Befehl »write #« (gleiche Syntax) verwenden. Andernfalls lassen sich die Werte nicht mehr über den Input # -Befehl korrekt einlesen.)
CHDIR	Aktuelles Verzeichnis wechseln	PUT	Datensatz schreiben
CLEAR	Variablen löschen und Dateien schließen (keine Zahlenangaben für Speicherplatz von Daten und Stack)	RANDOMIZE	Zufallszahlenstartwert festlegen
CLOSE	Eine oder alle Dateien schließen	READ	Datum lesen
CLS	Bildschirmfenster löschen	REM	Kommentar des Programmierers
COLOR	Farben setzen	RESET	Dateien schließen
DATA	Konstanten definieren (die maximale Anzahl der Data-Zeilen ist in der Variablen »maxdata« in Programmzeile 118 von »sbasic.bas« festgelegt)	RESTORE	Data-Zeiger setzen
DEF FN	Benutzerfunktion definieren (maximal vier Parameter, maximale Anzahl der Funktionen: 50)	RETURN	Rücksprung von einem Unterprogramm (befindet sich keine Adresse auf dem Stack, so erfolgt Sprung ins Ungewisse)
DEF SEG	Datensegment definieren	RMDIR	Unterverzeichnis entfernen
DEFINT, DEFSNG, DEFSTR	Variablentypen festlegen (keine dynamische Definition, das heißt, der Ausdruck »if test then defbl a-z else defsg a-z« ist nicht zulässig)	RSET	String rechtsbündig auffüllen
DIM	Speicherplatz für Felder reservieren (maximal vier Dimensionen, keine dynamische Dimensionierung in der Art »dim feld(anzahl)«)	RUN	Programm neu starten (Dateiangaben in der Art »run "name"« lassen sich nicht verwenden)
END	Programm beenden und Dateien schließen	SCREEN	Videomodus setzen (Grafikmodi und Farbangaben sind nicht erlaubt, aber das Arbeiten mit mehreren Bildschirmseiten ist möglich)
FIELD	Datensatzfelder festlegen	SHELL	Befehlsinterpreter oder Programm aufrufen
FILES	Inhaltsverzeichnis ausgeben	SOUND	Ton erzeugen
FOR ... NEXT ... STEP	Zählschleife (die maximale Anzahl der For-Next-Schleifen ist in der Variablen »maxfor« in Zeile 120 von »sbasic.bas« festgelegt)	STOP	Programm unter Angabe der Zeilennummer anhalten
GET	Datensatz lesen	SWAP	Variablen vertauschen
GOSUB	Unterprogramm aufrufen	SYSTEM	Programm beenden
GOTO	Zu neuer Adresse springen	VIEW PRINT	Bildschirmfenster festlegen (die Standardeinstellung ist »1 to 24«, der maximal zulässige Wert darf auch 25 sein)
IF ... THEN ... ELSE	Bedingte Verzweigung (die maximale Anzahl der If-Abfragen ist in der Variablen »maxif« in Programmzeile 116 von »sbasic.bas« festgelegt, auch verschachtelte Else-Konstrukte sind möglich)	WAIT	Portadresse abfragen
INPUT, LINE INPUT	Zeichenkette lesen (der Insert-Modus bleibt bei Cursor-Bewegungen aktiv; wortweises Springen ist nicht möglich; [Home] führt an den Anfang der Zeile; eine Fehlermeldung in der Art »Redo from Start« unterbleibt bei Fehleingaben; alle Zeichen werden vom Bildschirm übernommen; [End] ist nicht nötig; Cursor-Bewegungen über den linken Eingabebeginn beziehungsweise den rechten Bildschirmrand hinaus werden verhindert; die Eingabeumleitung funktioniert; bei der Eingabe des Gerätes »kybd:« erscheint der Cursor)	WHILE ... WEND	Abweisende Schleife (die maximale Anzahl der While-Wend-Schleifen ist in der Variablen »maxwhile« in Zeile 122 von »sbasic.bas« festgelegt)
KILL	Datei löschen (Wildcards dürfen nicht verwendet werden, in diesem Fall bietet sich als Ersatz »shell "del ..."« an)	WIDTH	Bildschirmbreite festlegen (die Angabe eines Kanals ist nicht erlaubt)
LET	Zuweisung	WRITE	Variablenwerte ausgeben
LOCATE	Cursor positionieren		
LSET	String linksbündig auffüllen	Funktionen:	
MKDIR	Unterverzeichnis anlegen	ABS()	Absoluter Betrag
NAME	Datei umbenennen	ASC()	Umwandlung eines ASCII-Zeichens in einen numerischen Wert
ON ... GOSUB	Eines von mehreren Unterprogrammen aufrufen	ATN()	Arkustangens
ON ... GOTO	Zu einer von mehreren Adressen springen	CHR\$()	Umwandlung eines numerischen Wertes in ein ASCII-Zeichen
OPEN	Datei öffnen (nur in der folgenden Form erlaubt: open "name" [for modus] as #nummer [len=satzlänge] das heißt, statt open "o", #nummer, "name" [,satzlänge] open "i", #nummer, "name" [,satzlänge] open "r", #nummer, "name" [,satzlänge] open "a", #nummer, "name" [,satzlänge] schreiben Sie: open "name" for output as #nummer [len=satzlänge] open "name" for input as #nummer [len=satzlänge] open "name" for random as #nummer [len=satzlänge])	CINT()	Umwandlung eines Fließkommawertes in einen Integerwert
		COMMAND\$()	Enthält die Parameter aus der Kommandozeile beim Aufruf des Basic-Programms
		COS()	Kosinus
		CSRLIN	Zeile, in der sich der Cursor befindet
		CVI()	Umwandlung eines Strings in einen Integerwert
		CVS()	Umwandlung eines Strings in einen Fließkommawert
		DAT\$()	Aktuelles Datum im amerikanischen Format
		EOF()	Dateiende erreicht
		EXP()	Exponentialfunktion »E ^ X« (die letzte Stelle ist unter Umständen nicht korrekt, weil keine zusätzlichen Stellen für die interne Verarbeitung vorgesehen sind)
		FIX()	Vorkommaanteil
		HEX\$()	Umwandlung eines numerischen Wertes in einen hexadezimalen String
		INKEY\$()	Gedrückte Taste
		INP()	Status eines Port

Anweisung	Erklärung (und Besonderheiten gegenüber GW-Basic)
INPUT\$()	Zeichen einlesen
INSTR()	String suchen
INT()	Ganzzahliger Wert
LEFT\$()	String verkürzen
LEN()	Stringlänge
LOG()	Natürlicher Logarithmus (die letzte Ziffer ist unter Umständen nicht korrekt, weil keine zusätzlichen Stellen für die interne Verarbeitung vorgesehen sind)
MID\$()	Teilstring (auch als Befehl erlaubt)
MKI\$()	Umwandlung eines Integerwertes in einen String
MKS\$()	Umwandlung eines Fließkommawertes in einen String
PEEK()	Wert einer Speicherzelle
POS()	Cursor-Spalte
RIGHT\$()	Teilstring
RND()	Zufallszahl
SCREEN()	Bildschirmzeichen oder Attribut
SGN()	Vorzeichen
SIN()	Sinus

Anweisung	Erklärung (und Besonderheiten gegenüber GW-Basic)
SPACE\$()	String mit Leerzeichen
SQR()	Quadratwurzel (Genauigkeitsprobleme, weil »sqr(a)« dem Ausdruck »exp(log(a)/2« entspricht)
STICK()	Joystick-Stellung (ein nicht angeschlossener Joystick liefert den Wert 508 zurück)
STRIG()	Joystick-Knöpfe (diese Funktion gibt zurück, welcher Joystick-Knopf gerade gedrückt ist; die Tastenzustände werden nicht zwischengespeichert (gepuffert); das Schalten mit »strig(n) on/off/stop« ist nicht möglich)
STR\$()	Umwandlung eines numerischen Wertes in einen String
STRING\$()	Sich wiederholende Zeichen
TAN()	Tangens
TIMES	Systemzeit
TIMER	Anzahl der seit Mitternacht verstrichenen Sekunden
VAL()	Umwandlung eines String in einen numerischen Wert
VARPTR()	Zeiger auf Variable (unter Umständen treten hier Probleme mit Strings auf, siehe dazu »call«)

Tabelle 2. Diese Befehle und Funktionen werden von SBasic unterstützt

Ändern Sie neben der jeweiligen Konstanten auch die Feldgrößen in den Dim-Anweisungen der folgenden Zeile, weil SBasic dynamische Felder in der Art

```
dim feld(max)
```

nicht unterstützt. Sichern Sie diese neue Version dann unter einem anderen Namen, und compilieren Sie sie mit der ersten Version.

Im Gegensatz zu GW-Basic, das alle Zahlen ins Fließkommaformat umwandelt, arbeitet SBasic mit echter Integer-Arithmetik. Als Beispiel können Sie einmal die Zahlen 2 und 32767 multiplizieren. Das Ergebnis lautet beim GW-Basic-Interpreter 65534, bei SBasic hingegen -2. Ist diese Form der Berechnung unerwünscht, so deklarieren Sie einfach eine der beiden Zahlen durch ein nachgestelltes Ausrufezeichen als

einfachgenaue Variable, die Berechnung erfolgt dadurch im Fließkommaformat.

Die logischen Operatoren »and«, »or«, »xor« und »not« werden von SBasic übrigens immer im Integer-Format ausgeführt, auch wenn eine Fließkommazahl angegeben ist.

Tabelle 1 zeigt die Operatoren von SBasic und ihre Prioritätsebenen, Tabelle 2 listet alle unter SBasic zulässigen Anweisungen mit Kurzbedeutung und Besonderheiten auf, und Tabelle 3 enthält die Bedeutung der Systemvariablen, die in jedem Compiler enthalten sind.

Wenn Sie übrigens die Laufzeitbibliothek »sbasic.asm« nicht abgetippt haben, können Sie zumindest einzelne Routinen daraus für Ihre Programmsammlung verwenden.

(Wolfgang Lorenz/ma)

Selbstverständlich werden DOS International und DOS Extra Sie als SBasic-Besitzer auch in Zukunft unterstützen. Bereits in dieser DOS Extra finden Sie einige Basic-Programme, die sich ohne Änderung mit SBasic compilieren lassen, unter anderem auch eine leistungsfähige Datenbank, und eine integrierte Entwicklungsumgebung steht kurz vor der Fertigstellung.

Wenn Sie selbst ein für den SBasic-Compiler geeignetes, interessantes Programm geschrieben haben, möchten wir Sie bitten, uns das Programm zuzusenden, damit wir es auch allen anderen Lesern zugänglich machen können.

Für den Fall, daß Sie sogar eigene Erweiterungen (beispielsweise für Grafikbefehle) geschrieben haben, gilt natürlich Entsprechendes. Und nicht nur ideell, sondern auch finanziell wird Ihre Arbeit selbstverständlich nicht umsonst sein.

Name	sbasic.bas
Sprache	GW-Basic
Grafikkarte	jede
Besonderheiten	Basic-Compiler

Name	sbasic.asm
Sprache	MASM ab Version 4.0, TASM
Grafikkarte	jede
Besonderheiten	Laufzeitbibliothek für SBasic

Die Belegung der internen Systemadressen 0 bis 47

Adresse	Bedeutung
0 bis 4	MS-DOS-Handles für offene Dateien, die Dateien 0 (für »scrn«, Pseudo-Handle -1) und 4 (»prn«) sind immer offen
5 bis 8	Spaltenpositionen des Cursor in den Dateien 1 bis 3 und beim Drucker
9	Aktuelles Bildschirmattribut
10	Flag, das angibt, ob das letzte Zeichen am rechten Rand ausgegeben wurde
11	Positionsnummer des letzten Zeichens in einer Bildschirmzeile
12 und 13	Obere und untere Grenze des Bildschirmfensters (für »view print«)
14	Ein-/Ausgabepuffer für Dateien
15 und 16	Cursor-Form beim Programmaufruf
17 bis 20	Nächstes zu lesendes Zeichen bei der Dateieingabe

Adresse	Bedeutung
21 und 22	Zwischenspeicher der Cursor-Form beim Befehl »input«
23 und 24	Zeiger auf aktuelles Data-Element
25 und 26	Aktuelles Datensegment (für »def seg«)
27 bis 29	Feldlängen der Dateien 1 bis 3 (für »field«)
30 bis 35	Adressen, an denen die Feldinformationen der Dateien 1 bis 3 beginnen
36 bis 37	Aktuelle Cursor-Form
38 bis 41	Letzter Zufallswert im einfachgenauen Format (für »rnd«)
42 und 43	Segmentadresse des PSP (»Program Segment Prefix«)
44	Aktuelle Oktave (für »play "O # "«)
45	Aktuelle Notenlänge (für »play "N # "«)
46	Aktuelles Musiktempo (»play "T # "«)
47	Aktuelle Tonlänge in Achtein (»play "MN/ML/MS"«)

Tabelle 3. Die Systemvariablen eines Compilers des SBasic-Compilers


```
[B22] 100 DEFINT A-Z
[RVG] 101 REM SBasic 1.0, (C) 1989 Redaktionsbüro Evert
s&Hagedorn, DOS International, Autor: Wolfgang Lo
renz
[QCE] 102 PRINT " SBasic 1.0, Compiler für GW-Basic, (C
) 1989 Everts&Hagedorn, DOS International
[YL9] 103 AUS$=SPACES(255):EINDAT$=COMMAND$(1):IF EINDA
T$<>" THEN 107
[KC1] 104 PRINT
[Q56] 105 INPUT "Quelldatei [.BAS]: ",EINDAT$
[7R3] 106 IF EINDAT$="" THEN END
[SB8] 107 IF INSTR(EINDAT$,".")=0 THEN EINDAT$=EINDAT$+
".BAS"
[Q0L3] 108 DEF FNLO(X)=X AND 255
[165] 109 DEF FNHI(X)=INT(X/256)-256*(X<0)
[1R4] 110 DEF FNI(X!)=X!+65536!*(X!>32767)
[384] 111 DEF FNRI(X)=X-65536!*(X<0)
[GF2] 112 MAXZEIL=1999
[545] 113 DIM ZEILNR(1999),ZEILADR(1999)
[K52] 114 MAXVAR=199
[637] 115 DIM VARNAME$(199),VARADR(199),VARDIM(199,4)
[HX1] 116 MAXIF=499
[HL4] 117 DIM IFADR(499),IFSTACK(499)
[K62] 118 MAXDATA=99
[Y84] 119 DIM DATLIN(99),DATADR(99)
[902] 120 MAXFOR=49
[NGF] 121 DIM FORNEXT(49),FORADR(49),FOREND(49),FORSTEP
(49),FORMOD(49),FORCHECK(49),FORADD(49),FORSTACK(
49)
[EE2] 122 MAXWHILE=49
[G16] 123 DIM WWEND(49),WCHECK(49),WSTACK(49)
[3ZA] 124 DIM TYPDEF(25),STACK(99),TYPEN(19),UPLN(199)
,UPADR(199),FIELDADR(49)
[V9B] 125 DIM STRPTR(99),ONADR(499),FUNNAME$(49),FUNTYP
(49),FUNPAR$(49),FUNPAR(49)
[P4A] 126 DIM FUNPARA(3,49),FUNPART(3,49),FUNANF(49),FU
NEND(49),FUNADR(49)
[K65] 127 OPEN "SBASIC.BIN" AS #3 LEN=128
[TM4] 128 FIELD #3,128 AS EIN3$:GET #3,1
[MMF] 129 GET #3,1:IF EOF(3) THEN PRINT:PRINT "Laufzeit
bibliothek SBASIC.BIN nicht gefunden":CLOSE:END
[C11] 130 I3=2
[WN3] 131 GOSUB 13500:WORT=EIN
[5Y5] 132 GOSUB 13500:WORT=FNI(WORT+EIN*256!)
[PC3] 133 IF WORT=-1 THEN 138
[2D6] 134 IF UPMAX THEN UPLN(UPMAX-1)=WORT-I
[3M1] 135 I=WORT
[RU2] 136 UPMAX=UPMAX+1
[1R1] 137 GOTO 131
[1A1] 138 VC!=48
[3Z1] 139 GOSUB 10000
[EC1] 140 PRINT
[KC1] 141 PRINT
[844] 142 PRINT "Bibliothek: ";RC!
[P14] 143 PRINT " Programm: ";PC!
[4P3] 144 PRINT " Stack: ";1024
[D34] 145 PRINT " Variablen: ";VC!
[U84] 146 PRINT " Strings: ";SC*16!
[RM7] 147 AUSDAT$=LEFT$(EINDAT$,INSTR(EINDAT$,".))+ "EX
E"
[XA5] 148 OPEN AUSDAT$ FOR OUTPUT AS #2
[5B5] 149 REM *** EXE-Header schreiben ***
[Y34] 150 WORT=&H5A4D:GOSUB 11043
[R26] 151 WORT=FNI(RC!+PC!) AND 511:GOSUB 11043
[DN6] 152 WORT=INT((512+RC!+PC!+511)/512):GOSUB 11043
[KA3] 153 WORT=0:GOSUB 11043
[UF3] 154 WORT=32:GOSUB 11043
[747] 155 WORT=FNI(256+INT((VC!+15)/16)+SC):GOSUB 11043
[P52] 156 GOSUB 11043
[RP6] 157 WORT=FNI(INT((RC!+PC!+15)/16)):GOSUB 11043
[VX6] 158 WORT=&H1000:GOSUB 11043:WORT=0:GOSUB 11043
[7P3] 159 WORT=RC!:GOSUB 11043
[TE6] 160 WORT=0:GOSUB 11043:WORT=28:GOSUB 11043
[NF3] 161 WORT=0:FOR I=1 TO 243
[952] 162 GOSUB 11043
[F81] 163 NEXT
[F76] 164 REM *** Unterprogramme schreiben ***
[641] 165 RC!=0
[YH1] 166 I=0:D=0
[DE6] 167 IF UPADR(I)=0 THEN D=D+UPLN(I):GOTO 174
[CF2] 168 UPADR(I)=RC!
[ZY2] 169 RC!=RC!+UPLN(I)
[RL2] 170 IF D=0 THEN 173
[KP4] 171 Z3=Z3+D:D=0:IF Z3<128 THEN 173
[5V5] 172 I3=I3+Z3:128:Z3=Z3 AND 127:GET #3,I3-1
[R1A] 173 FOR WORT=1 TO UPLN(I):GOSUB 13500:AUS=EIN:GO
SUB 11003:NEXT
[H71] 174 I=I+1
[1H3] 175 IF I<UPMAX THEN 167
[EN1] 176 CLOSE 3
[YD2] 177 FOR I=0 TO 25
[CA2] 178 TYPDEF(I)=0
[A91] 179 NEXT
[TF1] 180 PASS=1
[KY1] 181 GOSUB 10000
[ME4] 182 PRINT #2,LEFT$(AUS$,AUS$);
[1N1] 183 CLOSE 2
[AD1] 184 PRINT
[241] 185 END
```

```
[E37] 1000 REM *** Quelldatei zum Lesen öffnen ***
[EF4] 1001 OPEN EINDAT$ AS #1 LEN=128
[K93] 1002 FIELD #1,128 AS EIN$
[7K2] 1003 I1=1:EINZ=255
[452] 1004 GOSUB 9995
[CL8] 1005 IF EIN=254 THEN PRINT "Quelldatei geschützt"
:END
[5BA] 1006 IF EIN<255 THEN PRINT "Quelldatei nicht im T
oken-Format":END
[185] 1007 WORT=FNI(INT((PC!+RC!+15)/16)+256)
[HA1] 1008 PC!=0
[4K1] 1009 PRINT
[TL3] 1010 AUS=&HB8:GOSUB 11000
[Y42] 1011 GOSUB 11040
[FM3] 1012 AUS=&HB8:GOSUB 11000
[DD6] 1013 WORT=FNI(INT((VC!+15)/16)+SC):GOSUB 11040
[DB3] 1014 UP=&H0:GOSUB 11200
[TZ3] 1015 WORT=&H6C7:GOSUB 11040
[QT3] 1016 WORT=&H17:GOSUB 11040
[QY8] 1017 IF DATADR(0) THEN WORT=FNI(RC!+DATADR(0)) EL
SE WORT=0
[552] 1018 GOSUB 11040
[5T1] 1019 ZEILE=0
[WS1] 1020 IFCNT=0
[8Z1] 1021 FORCNT=0
[UB2] 1022 WHILECNT=0
[EX1] 1023 DATCNT=0
[X02] 1024 PTRCNT=0
[7A2] 1025 FIELDCNT=0
[MU1] 1026 ONCNT=0
[UB5] 1027 REM *** Zeilennummer lesen ***
[R42] 1028 GOSUB 10000
[P83] 1029 IF WORT THEN 1036
[H94] 1030 ZEILADR(ZEILE)=FNI(PC!)
[KS2] 1031 ZEILE=ZEILE+1
[MR2] 1032 ZEILEN=ZEILE
[SB3] 1033 UP=&H1:GOSUB 11200
[QNI] 1034 CLOSE 1
[9V1] 1035 RETURN
[5WC] 1036 GOSUB 10000:IF ZEILE>MAXZEIL THEN PRINT "Zu
viele Programmzeilen";GOTO 9990
[7U9] 1037 PRINT "Zeile: ";ZEILE+1;" (";MID$(STR$(FNR!(WO
RT)),2);";";CHR$(30))
[XZ9] 1038 ZEILNR(ZEILE)=WORT:ZEILADR(ZEILE)=FNI(PC!):Z
EILE=ZEILE+1
[CV5] 1039 REM *** Nächsten Befehl lesen ***
[G52] 1040 GOSUB 9995
[NU4] 1041 IF EIN=58 OR EIN=32 THEN 1039
[MB3] 1042 IF EIN=0 THEN 1028
[CE5] 1043 REM *** Variablenzuweisung ***
[5E5] 1044 IF EIN>=65 AND EIN<=90 THEN 1047
[EP3] 1045 IF EIN<>136 THEN 1055
[952] 1046 GOSUB 10030
[UF8] 1047 GOSUB 10070:IF CMDFLAG THEN PRINT "Zuweisung
an COMMAND$";:GOTO 9990
[9N2] 1048 LETTYP=TYP
[P48] 1049 IF EIN<>231 THEN PRINT "'=' erwartet";:GOTO
9990
[752] 1050 GOSUB 10030
[H32] 1051 GOSUB 6000
[1W4] 1052 NEUTYP=LETTYP:GOSUB 11600
[D52] 1053 GOSUB 13300
[HW1] 1054 GOTO 5000
[R22] 1055 REM *** PRINT ***
[NP3] 1056 IF EIN<>145 THEN 1092
[E52] 1057 GOSUB 10030
[ZB7] 1058 IF EIN<>35 THEN WORT=0:GOSUB 11080:GOTO 1062
[G52] 1059 GOSUB 10030
[W52] 1060 GOSUB 11900
[F23] 1061 X$=",";GOSUB 11800
[KP3] 1062 IF EIN<>206 THEN 1067
[E52] 1063 GOSUB 10030
[Q62] 1064 GOSUB 11900
[NE3] 1065 UP=&H5A:GOSUB 11200
[SW1] 1066 GOTO 1071
[8P3] 1067 IF EIN<>210 THEN 1072
[L52] 1068 GOSUB 10030
[562] 1069 GOSUB 11900
[NE3] 1070 UP=&H5B:GOSUB 11200
[A23] 1071 X$=")";GOSUB 11800
[U32] 1072 GOSUB 6000
[535] 1073 ON TYP GOTO 1075,1076,9980,1079
[TW1] 1074 GOTO 1080
[V53] 1075 UP=&H16:GOTO 1077
[AR1] 1076 UP=&H46
[K52] 1077 GOSUB 11200
[5V6] 1078 WORT=&H46FE:GOSUB 11040:AUS=0:GOSUB 11000
[1E3] 1079 UP=&H15:GOSUB 11200
[2L3] 1080 IF EIN<>59 THEN 1084
[M52] 1081 GOSUB 10030
[7T4] 1082 IF EIN=0 OR EIN=58 THEN 1090
[WW1] 1083 GOTO 1080
[RK3] 1084 IF EIN<>44 THEN 1088
[S52] 1085 GOSUB 10030
[2E3] 1086 UP=&H21:GOSUB 11200
[6X1] 1087 GOTO 1082
[L35] 1088 IF EIN<>0 AND EIN<>58 THEN 1062
[4E3] 1089 UP=&H14:GOSUB 11200
[CM3] 1090 AUS=&H58:GOSUB 11000
```



```
[YW1] 1091 GOTO 5000
[BA3] 1092 REM *** LPRINT ***
[RR3] 1093 IF EIN<>157 THEN 1097
[V52] 1094 GOSUB 10030
[4G3] 1095 WORT=4:GOSUB 11080
[5X1] 1096 GOTO 1062
[1D5] 1097 REM *** DEFINIT/SGN/DBL/STR ***
[C55] 1098 IF EIN<172 OR EIN>175 THEN 1114
[PP5] 1099 TYP=EIN-172:IF TYP=0 THEN TYP=4
[P42] 1100 GOSUB 10030
[7KA] 1101 IF EIN<65 OR EIN>90 THEN PRINT "Buchstabe er
wartet";:GOTO 9990
[6D2] 1102 WORT=EIN-65
[T42] 1103 GOSUB 10030
[HN3] 1104 IF EIN<>234 THEN 1112
[V42] 1105 GOSUB 10030
[ZS3] 1106 FOR I=WORT TO EIN-65
[AV2] 1107 TYPDEF(I)=TYP
[B91] 1108 NEXT
[Z42] 1109 GOSUB 10030
[AC3] 1110 IF EIN=44 THEN 1100
[1W1] 1111 GOTO 5000
[VF3] 1112 TYPDEF(WORT)=TYP
[PV1] 1113 GOTO 1110
[1H2] 1114 REM *** DIM ***
[EP3] 1115 IF EIN<>134 THEN 1149
[052] 1116 GOSUB 10030
[352] 1117 GOSUB 11400
[KU3] 1118 VARNAME$=VARNAME$+"("
[T63] 1119 IF PASS THEN 1124
[E04] 1120 FOR I=0 TO VARIABLE-1
[EUC] 1121 IF VARNAME$=VARNAME$(I) THEN PRINT "Feld ber
eits dimensioniert";:GOTO 9990
[D91] 1122 NEXT
[C35] 1123 VARNAME$(VARIABLE)=VARNAME$
[T13] 1124 X$="(":GOSUB 11800
[EV7] 1125 IF EIN>16 AND EIN<27 THEN WORT=EIN-17:GOTO 1
129
[6L7] 1126 IF EIN=15 THEN GOSUB 9995:WORT=EIN:GOTO 1129
[AY8] 1127 IF EIN=28 OR EIN=11 OR EIN=12 THEN GOSUB 100
00:GOTO 1129
[22A] 1128 PRINT "Variable Dimensionierung nicht erlaub
t";:GOTO 9990
[Z63] 1129 IF PASS THEN 1134
[N34] 1130 I=VARDIM(VARIABLE,0)+1
[F88] 1131 IF I>4 THEN PRINT "Maximal 4 Indizes";:GOTO
9990
[VT4] 1132 VARDIM(VARIABLE,I)=WORT+1
[3V3] 1133 VARDIM(VARIABLE,0)=I
[652] 1134 GOSUB 10030
[625] 1135 IF EIN=44 THEN GOSUB 10030:GOTO 1125
[223] 1136 X$=")":GOSUB 11800
[C73] 1137 IF PASS THEN 1147
[XE7] 1138 IF TYP=4 THEN VARLEN=16 ELSE VARLEN=2^TYP
[2A6] 1139 FOR I=VARDIM(VARIABLE,0) TO 1 STEP -1
[DK4] 1140 WORT=VARDIM(VARIABLE,I)
[NW4] 1141 VARDIM(VARIABLE,I)=VARLEN
[5Y3] 1142 VARLEN=VARLEN*WORT
[P91] 1143 NEXT
[PR9] 1144 IF TYP=4 THEN VARADR(VARIABLE)=SC:SC=SC+VARL
EN:GOTO 1146
[ELG] 1145 VARADR(VARIABLE)=FNI(VCI):VC!=VC!+VARLEN:IF
VCI>65535 THEN PRINT "Variablenspeicher voll";:G
OTO 9990
[3XD] 1146 VARIABLE=VARIABLE+1:IF VARIABLE>MAXVAR THEN
PRINT "Zu viele Variablen";:GOTO 9990
[7D3] 1147 IF EIN=44 THEN 1116
[MW1] 1148 GOTO 5000
[7X3] 1149 REM *** GOTO/GOSUB ***
[HA2] 1150 UP=EIN=141
[FV2] 1151 IF UP THEN 1154
[GD3] 1152 IF EIN=14 THEN 1156
[BP3] 1153 IF EIN<>137 THEN 1161
[E52] 1154 GOSUB 10030
[KU9] 1155 IF EIN<>14 THEN PRINT "Zeilennummer erwartet
";:GOTO 9990
[G52] 1156 GOSUB 13200
[TT5] 1157 IF UP THEN AUS=&HE8 ELSE AUS=&HE9
[852] 1158 GOSUB 11000
[B52] 1159 GOSUB 12000
[MW1] 1160 GOTO 5000
[CK2] 1161 REM *** REM ***
[XP3] 1162 IF EIN<>143 THEN 1166
[Y52] 1163 GOSUB 9995
[613] 1164 IF EIN THEN 1163
[TW1] 1165 GOTO 5000
[U93] 1166 REM *** RETURN ***
[MP3] 1167 IF EIN<>142 THEN 1171
[P52] 1168 GOSUB 10030
[GM3] 1169 AUS=&HC3:GOSUB 11000
[SW1] 1170 GOTO 5000
[PF2] 1171 REM *** IF ***
[7R3] 1172 IF EIN<>139 THEN 1186
[N52] 1173 GOSUB 10030
[Z32] 1174 GOSUB 6000
[635] 1175 ON TYP GOTO 1181,1176,9980,1180
[R04] 1176 WORT=&HC031:GOSUB 11040
[304] 1177 WORT=&H5050:GOSUB 11040
[NE3] 1178 UP=&H57:GOSUB 11200
```

```
[5X1] 1179 GOTO 1181
[ME3] 1180 UP=&H66:GOSUB 11200
[7D8] 1181 IF EIN<>205 THEN PRINT "THEN erwartet";:GOTO
9990
[PZ4] 1182 ADR=IFADR(IFCNT):GOSUB 12500
[HS4] 1183 IFSTACK(IP)=IFCNT:IP=IP+1
[AED] 1184 IFCNT=IFCNT+1:IF IFCNT>MAXIF THEN PRINT "Zu
viele IF/THEN-Schleifen";:GOTO 9990
[1X1] 1185 GOTO 5000
[MP2] 1186 REM *** ELSE ***
[LR3] 1187 IF EIN<>161 THEN 1195
[Y52] 1188 GOSUB 10030
[6N3] 1189 AUS=&HE9:GOSUB 11000
[ZR5] 1190 WORT=ZEILADR(ZEILE):GOSUB 12000
[9S1] 1191 IP=IP-1
[DF7] 1192 IF IP<0 THEN PRINT "ELSE ohne IF";:GOTO 9990
[1T4] 1193 IFADR(IFSTACK(IP))=FNI(PC!)
[ZW1] 1194 GOTO 1041
[5L2] 1195 REM *** FOR ***
[SP3] 1196 IF EIN<>130 THEN 1261
[162] 1197 GOSUB 10030
[L5C] 1198 IF PASS THEN IF FORNEXT(FORCNT)=0 THEN PRINT
"FOR ohne NEXT";:GOTO 9990
[Y52] 1199 GOSUB 12100
[BN2] 1200 FORTYP=TYP
[U03] 1201 FORADR(FORCNT)=ADR
[548] 1202 IF EIN<>231 THEN PRINT "'=' erwartet";:GOTO
9990
[W42] 1203 GOSUB 10030
[632] 1204 GOSUB 6000
[0W4] 1205 NEUTYP=FORTYP:GOSUB 11600
[BT3] 1206 ADR=FORADR(FORCNT)
[152] 1207 GOSUB 12300
[9A8] 1208 IF EIN<>204 THEN PRINT "TO erwartet";:GOTO 9
990
[252] 1209 GOSUB 10030
[632] 1210 GOSUB 6000
[0W4] 1211 NEUTYP=FORTYP:GOSUB 11600
[Y99] 1212 IF PASS=0 THEN FOREND(FORCNT)=FNI(VCI):VC!=V
CI+2^FORTYP
[0G5] 1213 ADR=FOREND(FORCNT):GOSUB 12300
[BH3] 1214 IF EIN=207 THEN 1218
[YE3] 1215 WORT=1:GOSUB 11080
[FF1] 1216 TYP=1
[0W1] 1217 GOTO 1220
[552] 1218 GOSUB 10030
[F32] 1219 GOSUB 6000
[3W4] 1220 NEUTYP=FORTYP:GOSUB 11600
[20A] 1221 IF PASS=0 THEN FORSTEP(FORCNT)=FNI(VCI):VC!=
VC!+2^FORTYP
[PN5] 1222 ADR=FORSTEP(FORCNT):GOSUB 12300
[852] 1223 GOSUB 12400
[MY4] 1224 ON FORTYP GOTO 1225,1226,9980
[G53] 1225 UP=&H1A:GOTO 1227
[3R1] 1226 UP=&H59
[452] 1227 GOSUB 11200
[104] 1228 WORT=&H8058:GOSUB 11040
[Z23] 1229 WORT=&H5E4:GOSUB 11040
[HM3] 1230 AUS=&H2E:GOSUB 11000
[G04] 1231 WORT=&H2688:GOSUB 11040
[CS7] 1232 WORT=FNI(RCI+FNR!(FORMOD(FORCNT))):GOSUB 110
40
[FM3] 1233 AUS=&HE9:GOSUB 11000
[SY5] 1234 WORT=FORCHECK(FORCNT):GOSUB 12000
[MB4] 1235 FORADD(FORCNT)=FNI(PC!)
[ZG5] 1236 ADR=FORADR(FORCNT):GOSUB 12400
[2P5] 1237 ADR=FORSTEP(FORCNT):GOSUB 12400
[UY4] 1238 ON FORTYP GOTO 1239,1240,9980
[223] 1239 UP=&H8:GOTO 1241
[0R1] 1240 UP=&H48
[652] 1241 GOSUB 11200
[XG5] 1242 ADR=FORADR(FORCNT):GOSUB 12300
[RL4] 1243 FORCHECK(FORCNT)=FNI(PC!)
[1H5] 1244 ADR=FORADR(FORCNT):GOSUB 12400
[GG5] 1245 ADR=FOREND(FORCNT):GOSUB 12400
[VV3] 1246 WORT=&HEB:GOSUB 11040
[MM4] 1247 FORMOD(FORCNT)=FNI(PC!-1)
[3Z4] 1248 ON FORTYP GOTO 1249,1250,9980
[Z13] 1249 UP=&H6:GOTO 1251
[6R1] 1250 UP=&H56
[A52] 1251 GOSUB 11200
[S14] 1252 WORT=&H3EB:GOSUB 11040
[BZ4] 1253 ON FORTYP GOTO 1254,1255,9980
[423] 1254 UP=&H5:GOTO 1256
[AR1] 1255 UP=&H55
[F52] 1256 GOSUB 11200
[6R5] 1257 ADR=FORNEXT(FORCNT):GOSUB 12500
[845] 1258 FORSTACK(FP)=FORCNT:FP=FP+1
[EME] 1259 FORCNT=FORCNT+1:IF FORCNT>MAXFOR THEN PRINT
"Zu viele FOR/NEXT-Schleifen";:GOTO 9990
[RW1] 1260 GOTO 5000
[CR2] 1261 REM *** NEXT ***
[MP3] 1262 IF EIN<>131 THEN 1273
[M52] 1263 GOSUB 10030
[MR1] 1264 FP=FP-1
```

Listing 1. Der Quelltext des SBasic-Compilers »sbasic.bas«


```
[LN7] 1265 IF FP<0 THEN PRINT "NEXT ohne FOR";:GOTO 999
[YM3] 1266 AUS=&HE9:GOSUB 11000
[EK6] 1267 WORT=FORADD(FORSTACK(FP)):GOSUB 12000
[HE5] 1268 FORNEXT(FORSTACK(FP))=FNI(PCI)
[BV4] 1269 IF EIN<65 OR EIN>90 THEN 5000
[G52] 1270 GOSUB 12100
[4E3] 1271 IF EIN=44 THEN 1263
[XW1] 1272 GOTO 5000
[9K2] 1273 REM *** END ***
[DR3] 1274 IF EIN<>129 THEN 1278
[U52] 1275 GOSUB 10030
[HC3] 1276 UP=&H1:GOSUB 11200
[2X1] 1277 GOTO 5000
[4Y2] 1278 REM *** WHILE ***
[GR3] 1279 IF EIN<>177 THEN 1290
[662] 1280 GOSUB 9995
[H04] 1281 IF EIN<>233 THEN STOP
[V52] 1282 GOSUB 10030
[09C] 1283 IF PASS THEN IF WWEND(WHILECNT)=0 THEN PRINT
    "WHILE ohne WEND";:GOTO 999
[UP4] 1284 WCHECK(WHILECNT)=FNI(PCI)
[F62] 1285 GOSUB 11900
[7N5] 1286 ADR=WWEND(WHILECNT):GOSUB 12500
[SA5] 1287 WSTACK(WP)=WHILECNT:WP=WP+1
[NBG] 1288 WHILECNT=WHILECNT+1:IF WHILECNT>MAXWHILE THE
    N PRINT "Zu viele WHILE/WEND-Schleifen";:GOTO 999
[8X1] 1289 GOTO 5000
[8R2] 1290 REM *** WEND ***
[CS3] 1291 IF EIN<>178 THEN 1299
[Z52] 1292 GOSUB 10030
[MU1] 1293 WP=WP-1
[068] 1294 IF WP<0 THEN PRINT "WEND ohne WHILE";:GOTO 9
    990
[9N3] 1295 AUS=&HE9:GOSUB 11000
[NB6] 1296 WORT=WCHECK(WSTACK(WP)):GOSUB 12000
[PR4] 1297 WWEND(WSTACK(WP))=FNI(PCI)
[BX1] 1298 GOTO 5000
[SK2] 1299 REM *** CLS ***
[CP3] 1300 IF EIN<>192 THEN 1304
[X42] 1301 GOSUB 10030
[9D3] 1302 UP=&H17:GOSUB 11200
[5W1] 1303 GOTO 5000
[573] 1304 REM *** LOCATE ***
[RN3] 1305 IF EIN<>202 THEN 1325
[Z52] 1306 GOSUB 10030
[M52] 1307 GOSUB 11900
[PM5] 1308 IF TYP=0 THEN UP=&H5C:GOSUB 11200
[XK4] 1309 IF EIN=44 THEN GOSUB 10030
[H52] 1310 GOSUB 11900
[PM5] 1311 IF TYP=0 THEN UP=&H5D:GOSUB 11200
[SD3] 1312 UP=&H1B:GOSUB 11200
[RH3] 1313 IF EIN<>44 THEN 5000
[452] 1314 GOSUB 10030
[P52] 1315 GOSUB 11900
[1N5] 1316 IF TYP=0 THEN UP=&H5F:GOSUB 11200
[ZK4] 1317 IF EIN=44 THEN GOSUB 10030
[T52] 1318 GOSUB 11900
[4L5] 1319 IF TYP=0 THEN UP=&H60:GOSUB 11200
[WK4] 1320 IF EIN=44 THEN GOSUB 10030
[P52] 1321 GOSUB 11900
[4L5] 1322 IF TYP=0 THEN UP=&H61:GOSUB 11200
[GE3] 1323 UP=&H5E:GOSUB 11200
[EW1] 1324 GOTO 5000
[0Y2] 1325 REM *** COLOR ***
[SP3] 1326 IF EIN<>191 THEN 1341
[B52] 1327 GOSUB 10030
[X52] 1328 GOSUB 11900
[EL5] 1329 IF TYP=0 THEN UP=&H62:GOSUB 11200
[6H5] 1330 IF EIN<>44 THEN TYP=0:GOTO 1333
[C23] 1331 X$="," :GOSUB 11800
[V52] 1332 GOSUB 11900
[FL5] 1333 IF TYP=0 THEN UP=&H63:GOSUB 11200
[3E3] 1334 UP=&H1C:GOSUB 11200
[1K3] 1335 IF EIN<>44 THEN 5000
[E52] 1336 GOSUB 10030
[062] 1337 GOSUB 11900
[LD3] 1338 IF TYP=0 THEN 5000
[SE3] 1339 UP=&H87:GOSUB 11200
[KW1] 1340 GOTO 5000
[CY2] 1341 REM *** WIDTH ***
[1R3] 1342 IF EIN<>160 THEN 1347
[F52] 1343 GOSUB 10030
[162] 1344 GOSUB 11900
[9E3] 1345 UP=&H1D:GOSUB 11200
[SW1] 1346 GOTO 5000
[UY3] 1347 REM *** VIEW PRINT ***
[5R3] 1348 IF EIN<>254 THEN 1364
[Z62] 1349 GOSUB 9995
[FR3] 1350 IF EIN<>156 THEN 1587
[H52] 1351 GOSUB 10030
[0T8] 1352 IF EIN<>145 THEN PRINT "PRINT erwartet";:GOT
    O 9990
[L52] 1353 GOSUB 10030
[562] 1354 GOSUB 11900
[GE3] 1355 IF TYP=0 THEN 1361
[YA8] 1356 IF EIN<>204 THEN PRINT "TO erwartet";:GOTO 9
    990
```

```
[R52] 1357 GOSUB 10030
[962] 1358 GOSUB 11900
[KE3] 1359 UP=&H1E:GOSUB 11200
[UW1] 1360 GOTO 5000
[HF3] 1361 WORT=1:GOSUB 11080
[DH3] 1362 WORT=23:GOSUB 11080
[NX1] 1363 GOTO 1359
[ZS2] 1364 REM *** OPEN ***
[YR3] 1365 IF EIN<>186 THEN 1395
[U52] 1366 GOSUB 10030
[442] 1367 GOSUB 6000
[G19] 1368 IF TYP<>4 THEN PRINT "Dateiname erwartet";:G
    OTO 9990
[2T5] 1369 IF EIN<>130 THEN WORT=2:GOTO 1381
[S52] 1370 GOSUB 10030
[2B7] 1371 IF EIN=133 THEN GOSUB 10030:WORT=0:GOTO 1381
[861] 1372 X$=""
[2W4] 1373 IF EIN<65 OR EIN>90 THEN 1377
[FT2] 1374 X$=X$+CHR$(EIN)
[A62] 1375 GOSUB 9995
[EX1] 1376 GOTO 1373
[3C6] 1377 IF X$="OUTPUT" THEN WORT=1:GOTO 1381
[D86] 1378 IF X$="RANDOM" THEN WORT=2:GOTO 1381
[986] 1379 IF X$="APPEND" THEN WORT=3:GOTO 1381
[XAA] 1380 PRINT "INPUT, OUTPUT, RANDOM oder APPEND erw
    artet";:GOTO 9990
[L62] 1381 GOSUB 11080
[662] 1382 GOSUB 10050
[K62] 1383 GOSUB 12900
[TL4] 1384 IF EIN=35 THEN GOSUB 10030
[K62] 1385 GOSUB 11900
[BY7] 1386 IF EIN<>255 THEN WORT=128:GOSUB 11080:GOTO 1
    393
[G62] 1387 GOSUB 9995
[AB8] 1388 IF EIN<>146 THEN PRINT "LEN erwartet";:GOTO
    9990
[562] 1389 GOSUB 10030
[A58] 1390 IF EIN<>231 THEN PRINT "'=' erwartet";:GOTO
    9990
[162] 1391 GOSUB 10030
[L62] 1392 GOSUB 11900
[XE3] 1393 UP=&H1F:GOSUB 11200
[AX1] 1394 GOTO 5000
[MX2] 1395 REM *** CLOSE ***
[SR3] 1396 IF EIN<>187 THEN 1404
[762] 1397 GOSUB 10030
[5H8] 1398 IF EIN=58 OR EIN=0 THEN UP=&H26:GOSUB 11200:
    GOTO 5000
[2M4] 1399 IF EIN=35 THEN GOSUB 10030
[G52] 1400 GOSUB 11900
[8D3] 1401 UP=&H20:GOSUB 11200
[R52] 1402 IF EIN=44 THEN GOSUB 10030:GOTO 1399
[8W1] 1403 GOTO 5000
[MW3] 1404 REM *** LINE INPUT ***
[WP3] 1405 IF EIN<>176 THEN 1417
[552] 1406 GOSUB 10030
[GC9] 1407 IF EIN<>133 THEN PRINT "Unbekannter Befehl";
    :GOTO 9990
[752] 1408 GOSUB 10030
[5C7] 1409 IF EIN<>35 THEN WORT=0:GOSUB 11080:GOTO 1413
[352] 1410 GOSUB 10030
[N52] 1411 GOSUB 11900
[823] 1412 X$="," :GOSUB 11800
[629] 1413 IF EIN=34 THEN GOSUB 6000:UP=&H15:GOSUB 1120
    0:GOSUB 10030
[PHB] 1414 GOSUB 10070:IF TYP<>4 THEN PRINT "Stringvari
    able erwartet";:GOTO 9990
[KD3] 1415 UP=&H22:GOSUB 11200
[FW1] 1416 GOTO 5000
[522] 1417 REM *** INPUT ***
[MP3] 1418 IF EIN<>133 THEN 1437
[C52] 1419 GOSUB 10030
[CK3] 1420 IF EIN<>34 THEN 1427
[0F3] 1421 WORT=0:GOSUB 11080
[K32] 1422 GOSUB 6000
[LD3] 1423 UP=&H15:GOSUB 11200
[N25] 1424 IF EIN=44 THEN GOSUB 10030:GOTO 1434
[506] 1425 IF EIN=59 THEN GOSUB 10030:GOTO 1433
[ZL6] 1426 PRINT ",'" oder ',' erwartet";:GOTO 9990
[EK3] 1427 IF EIN<>35 THEN 1432
[F52] 1428 GOSUB 10030
[162] 1429 GOSUB 11900
[E23] 1430 X$="," :GOSUB 11800
[SW1] 1431 GOTO 1434
[5F3] 1432 WORT=0:GOSUB 11080
[TD3] 1433 UP=&H23:GOSUB 11200
[T52] 1434 GOSUB 12600
[WD3] 1435 UP=&H30:GOSUB 11200
[RW1] 1436 GOTO 5000
[1P2] 1437 REM *** READ ***
[RP3] 1438 IF EIN<>135 THEN 1444
[M52] 1439 GOSUB 10030
[GG3] 1440 WORT=-1:GOSUB 11080
[U52] 1441 GOSUB 12600
[6M3] 1442 AUS=&H58:GOSUB 11000
[SW1] 1443 GOTO 5000
[NM2] 1444 REM *** DATA ***
[6R3] 1445 IF EIN<>132 THEN 1459
[N52] 1446 GOSUB 10030
```



```
[XM3] 1447 AUS=&HE9:GOSUB 11000
[WR5] 1448 WORT=ZEILADR(ZEILE):GOSUB 12000
[SA4] 1449 DATADR(DATCNT)=FNI(PC!)
[UL5] 1450 DATLIN(DATCNT)=ZEILNR(ZEILE-1)
[5HD] 1451 DATCNT=DATCNT+1:IF DATCNT>MAXDATA THEN PRINT
      "Zu viele DATA-Zeilen";:GOTO 9990
[DC3] 1452 IF EIN=0 THEN 1456
[5K3] 1453 AUS=EIN:GOSUB 11000
[462] 1454 GOSUB 9995
[2X1] 1455 GOTO 1452
[TB3] 1456 AUS=26:GOSUB 11000
[W0A] 1457 WORT=DATADR(DATCNT):IF WORT THEN WORT=FNI(RC
      !+FNR!(WORT))
[CM3] 1458 GOSUB 11040:GOTO 5000
[1C3] 1459 REM *** RESTORE ***
[RP3] 1460 IF EIN<>140 THEN 1471
[S52] 1461 GOSUB 10030
[UR1] 1462 WORT=0
[AA6] 1463 IF EIN=14 THEN GOSUB 10000:GOSUB 10030
[BB3] 1464 I=0:X!=FNR!(WORT)
[L54] 1465 IF DATADR(I)=0 THEN 1467
[727] 1466 IF FNR!(DATLIN(I))<X! THEN I=I+1:GOTO 1465
[V04] 1467 WORT=&H6C7:GOSUB 11040
[2U3] 1468 WORT=&H17:GOSUB 11040
[7B5] 1469 WORT=RC!+DATADR(I):GOSUB 11040
[1X1] 1470 GOTO 5000
[W33] 1471 REM *** DEF FN ***
[XP3] 1472 IF EIN<>151 THEN 1541
[Y52] 1473 GOSUB 10030
[7R3] 1474 IF EIN<>209 THEN 1528
[062] 1475 GOSUB 10030
[362] 1476 GOSUB 11400
[G21] 1477 I=0
[CT3] 1478 IF I=FUNNR THEN 1483
[UV5] 1479 IF VARNAME$=FUNNAME$(I) THEN 1482
[FA1] 1480 I=I+1
[0Y1] 1481 GOTO 1478
[8FA] 1482 IF PASS=0 THEN PRINT "Funktion bereits defin
      iert";:GOTO 9990
[UG1] 1483 FUN=I
[XW3] 1484 FUNNAME$(I)=VARNAME$
[TY2] 1485 FUNTYP(I)=TYP
[KW3] 1486 IF EIN<>40 THEN STOP
[662] 1487 GOSUB 10030
[962] 1488 GOSUB 11400
[TT4] 1489 FUNPAR$(FUNCNT)=VARNAME$
[B04] 1490 FUNPAR(FUNCNT)=TYP
[B83] 1491 IF PASS THEN 1494
[S49] 1492 IF TYP=4 THEN FUNPARA(I,FUNCNT)=SC:SC=SC+16:
      GOTO 1494
[PV6] 1493 FUNPARA(I,FUNCNT)=FNI(VCI):VCI=VCI+2:TYP
[574] 1494 FUNPART(I,FUNCNT)=TYP
[S46] 1495 FUNADR(FUNCNT)=FUNPARA(I,FUNCNT)
[3KC] 1496 FUNCNT=FUNCNT+1:IF FUNCNT>49 THEN PRINT "Zu
      viele Funktionen";:GOTO 9990
[ZD3] 1497 IF EIN=41 THEN 1502
[U39] 1498 IF EIN<>44 THEN PRINT "',' oder ')' erwartet
      ";:GOTO 9990
[BK9] 1499 IF FUNCNT=4 THEN PRINT "Maximal 4 Parameter"
      ";:GOTO 9990
[252] 1500 GOSUB 10030
[5X1] 1501 GOTO 1488
[452] 1502 GOSUB 10030
[F48] 1503 IF EIN<>231 THEN PRINT "'=' erwartet";:GOTO
      9990
[652] 1504 GOSUB 10030
[EM3] 1505 AUS=&HE9:GOSUB 11000
[CT4] 1506 WORT=FUNEND(I):GOSUB 12000
[LC3] 1507 FUNANF(I)=FNI(PC!)
[LM3] 1508 AUS=&H5B:GOSUB 11000
[BR4] 1509 FOR J=FUNCNT-1 TO 0 STEP -1
[NX2] 1510 TYP=FUNPAR(J)
[PS2] 1511 ADR=FUNADR(J)
[Y06] 1512 IF TYP<4 THEN GOSUB 12300:GOTO 1515
[P52] 1513 GOSUB 12700
[SD3] 1514 UP=&H32:GOSUB 11200
[S91] 1515 NEXT
[RL3] 1516 AUS=&H53:GOSUB 11000
[P32] 1517 GOSUB 6000
[2M5] 1518 NEUTYP=FUNTYP(FUN):GOSUB 11600
[F02] 1519 FUNCNT=0
[C96] 1520 ON FUNTYP(FUN) GOTO 1521,1522,9980,1523
[953] 1521 UP=&H33:GOTO 1524
[R53] 1522 UP=&H58:GOTO 1524
[WP1] 1523 UP=&H34
[A52] 1524 GOSUB 11200
[173] 1525 FUNEND(FUN)=PC!
[X85] 1526 IF PASS=0 THEN FUNNR=FUNNR+1
[RW1] 1527 GOTO 5000
[K53] 1528 REM *** DEF SEG ***
[4E2] 1529 X$=CHR$(EIN)
[RE7] 1530 GOSUB 9995:X$=X$+CHR$(EIN):IF LEN(X$)<3 THEN
      1530
[7R9] 1531 IF X$<>"SEG" THEN PRINT "'FN' oder 'SEG' erw
     artet";:GOTO 9990
[G52] 1532 GOSUB 10030
[2R3] 1533 IF EIN<>231 THEN 1539
[K52] 1534 GOSUB 10030
[462] 1535 GOSUB 11900
```

```
[C14] 1536 WORT=&H68F:GOSUB 11040
[YG3] 1537 WORT=25:GOSUB 11040
[WW1] 1538 GOTO 5000
[R24] 1539 WORT=&H18C:GOSUB 11040
[8X1] 1540 GOTO 1537
[GR2] 1541 REM *** POKE ***
[DR3] 1542 IF EIN<>152 THEN 1549
[N52] 1543 GOSUB 10030
[762] 1544 GOSUB 11900
[U23] 1545 X$=",";:GOSUB 11800
[962] 1546 GOSUB 11900
[AE3] 1547 UP=&H35:GOSUB 11200
[0X1] 1548 GOTO 5000
[8N2] 1549 REM *** CALL ***
[WR3] 1550 IF EIN<>179 THEN 1567
[R52] 1551 GOSUB 10030
[U52] 1552 GOSUB 11400
[S52] 1553 GOSUB 12104
[V22] 1554 CTYP=TYP
[2X1] 1555 CADR=ADR
[GR3] 1556 IF EIN<>40 THEN 1561
[X52] 1557 GOSUB 10030
[E62] 1558 GOSUB 12800
[R06] 1559 IF EIN=44 THEN GOSUB 10030:GOTO 1558
[M23] 1560 X$=",";:GOSUB 11800
[642] 1561 TYP=CTYP
[ZW1] 1562 ADR=CADR
[162] 1563 GOSUB 12400
[NV3] 1564 NEUTYP=1:GOSUB 11600
[Z33] 1565 UP=&H3D:GOSUB 11200
[6X1] 1566 GOTO 5000
[RW3] 1567 REM *** MID$(BEFEHL) ***
[XP3] 1568 IF EIN<>255 THEN 2000
[G62] 1569 GOSUB 9995
[GR3] 1570 IF EIN<>131 THEN 1587
[Z52] 1571 GOSUB 10030
[R23] 1572 X$=",";:GOSUB 11800
[H62] 1573 GOSUB 10070
[3L3] 1574 IF TYP<>4 THEN 9980
[633] 1575 X$=",";:GOSUB 11800
[N62] 1576 GOSUB 11900
[7T7] 1577 IF EIN<>44 THEN WORT=255:GOSUB 11080:GOTO 15
      80
[662] 1578 GOSUB 10030
[S62] 1579 GOSUB 11900
[W23] 1580 X$=",";:GOSUB 11800
[D58] 1581 IF EIN<>231 THEN PRINT "'=' erwartet";:GOTO
      9990
[462] 1582 GOSUB 10030
[E42] 1583 GOSUB 6000
[7L3] 1584 IF TYP<>4 THEN 9980
[8F3] 1585 UP=&H3E:GOSUB 11200
[EX1] 1586 GOTO 5000
[UW2] 1587 REM *** FIELD ***
[WP3] 1588 IF EIN<>130 THEN 1611
[B62] 1589 GOSUB 10030
[ZL4] 1590 IF EIN=35 THEN GOSUB 10030
[S62] 1591 GOSUB 11900
[BG8] 1592 IF PASS=0 THEN FIELDADR(FIELDCNT)=FNI(VCI):V
      CI=VCI+1
[7C6] 1593 WORT=FIELDADR(FIELDCNT):GOSUB 11080
[VA2] 1594 FIELDADR=0
[E33] 1595 X$=",";:GOSUB 11800
[X62] 1596 GOSUB 11900
[0Y7] 1597 ADR=FNI(FNR!(FIELDADR(FIELDCNT))+1+FIELDADR*
      4)
[F62] 1598 GOSUB 12300
[272] 1599 GOSUB 12900
[N52] 1600 GOSUB 10070
[7K3] 1601 IF TYP<>4 THEN 9980
[LF1] 1602 TYP=1
[3X7] 1603 ADR=FNI(FNR!(FIELDADR(FIELDCNT))+3+FIELDADR*
      4)
[A52] 1604 GOSUB 12300
[PR3] 1605 FIELDADR=FIELDADR+1
[654] 1606 IF PASS=0 THEN VCI=VCI+4
[HE3] 1607 IF EIN=44 THEN 1595
[FR4] 1608 WORT=FIELDADR:GOSUB 11080
[YD3] 1609 UP=&H40:GOSUB 11200
[4VF] 1610 FIELDCNT=FIELDCNT+1:IF FIELDADR<50 THEN 5000
      ELSE PRINT "Zu viele FIELD-Definitionen";:GOTO 9
      990
[DH2] 1611 REM *** GET ***
[3P3] 1612 IF EIN<>137 THEN 1620
[C52] 1613 GOSUB 10030
[6L4] 1614 IF EIN=35 THEN GOSUB 10030
[Z52] 1615 GOSUB 11900
[K23] 1616 X$=",";:GOSUB 11800
[N62] 1617 GOSUB 11950
[2E3] 1618 UP=&H41:GOSUB 11200
[SW1] 1619 GOTO 5000
[6S2] 1620 REM *** LSET ***
[HP3] 1621 IF EIN<>133 THEN 1625
[952] 1622 GOSUB 13000
[2E3] 1623 UP=&H42:GOSUB 11200
```

Listing 1. Der Quelltext des SBasic-Compilers »sbasic.bas«


```
[RW1] 1624 GOTO 5000
[XS2] 1625 REM *** RSET ***
[AP3] 1626 IF EIN<>134 THEN 1630
[E52] 1627 GOSUB 10030
[8E3] 1628 UP=&H43:GOSUB 11200
[WW1] 1629 GOTO 5000
[FM2] 1630 REM *** PUT ***
[6R3] 1631 IF EIN<>136 THEN 1639
[L52] 1632 GOSUB 10030
[DL4] 1633 IF EIN=35 THEN GOSUB 10030
[662] 1634 GOSUB 11900
[T23] 1635 X$=",";GOSUB 11800
[W62] 1636 GOSUB 11950
[CE3] 1637 UP=&H44:GOSUB 11200
[ZW1] 1638 GOTO 5000
[NN4] 1639 REM *** Zuweisung TIMES ***
[8R3] 1640 IF EIN<>142 THEN 1648
[P52] 1641 GOSUB 10030
[158] 1642 IF EIN<>231 THEN PRINT "'=' erwartet";:GOTO
9990
[S52] 1643 GOSUB 10030
[242] 1644 GOSUB 6000
[VK3] 1645 IF TYP<>4 THEN 9980
[XE3] 1646 UP=&H76:GOSUB 11200
[2X1] 1647 GOTO 5000
[9Y2] 1648 REM *** SHELL ***
[LR3] 1649 IF EIN<>154 THEN 1656
[T52] 1650 GOSUB 10030
[342] 1651 GOSUB 6000
[Z67] 1652 IF TYP=0 THEN UP=&H7B:GOSUB 11200:GOTO 1654
[XK3] 1653 IF TYP<>4 THEN 9980
[CF3] 1654 UP=&H7C:GOSUB 11200
[4X1] 1655 GOTO 5000
[GY2] 1656 REM *** MKDIR ***
[CR3] 1657 IF EIN<>152 THEN 1663
[162] 1658 GOSUB 10030
[B42] 1659 GOSUB 6000
[YK3] 1660 IF TYP<>4 THEN 9980
[GF3] 1661 UP=&H7F:GOSUB 11200
[5X1] 1662 GOTO 5000
[822] 1663 REM *** RMDIR ***
[7R3] 1664 IF EIN<>153 THEN 1670
[262] 1665 GOSUB 10030
[C42] 1666 GOSUB 6000
[5L3] 1667 IF TYP<>4 THEN 9980
[5F3] 1668 UP=&H80:GOSUB 11200
[CX1] 1669 GOTO 5000
[8X2] 1670 REM *** CHDIR ***
[VR3] 1671 IF EIN<>151 THEN 1677
[362] 1672 GOSUB 10030
[D42] 1673 GOSUB 6000
[6L3] 1674 IF TYP<>4 THEN 9980
[7F3] 1675 UP=&H81:GOSUB 11200
[DX1] 1676 GOTO 5000
[XY2] 1677 REM *** RESET ***
[MR3] 1678 IF EIN<>138 THEN 1682
[A62] 1679 GOSUB 10030
[LE3] 1680 UP=&H26:GOSUB 11200
[CX1] 1681 GOTO 5000
[PS2] 1682 REM *** PLAY ***
[BS3] 1683 IF EIN<>147 THEN 1689
[962] 1684 GOSUB 10030
[L42] 1685 GOSUB 6000
[CL3] 1686 IF TYP<>4 THEN 9980
[NF3] 1687 UP=&H89:GOSUB 11200
[ LX1] 1688 GOTO 5000
[LR2] 1689 REM *** KILL ***
[ZR3] 1690 IF EIN<>135 THEN 1696
[A62] 1691 GOSUB 10030
[M42] 1692 GOSUB 6000
[DL3] 1693 IF TYP<>4 THEN 9980
[OG3] 1694 UP=&H8C:GOSUB 11200
[MX1] 1695 GOTO 5000
[6Y2] 1696 REM *** FILES ***
[FR3] 1697 IF EIN<>129 THEN 1704
[H62] 1698 GOSUB 10030
[V42] 1699 GOSUB 6000
[1N6] 1700 IF TYP=0 THEN UP=&H7B:GOSUB 11200:TYP=4
[AK3] 1701 IF TYP<>4 THEN 9980
[YE3] 1702 UP=&H8D:GOSUB 11200
[HW1] 1703 GOTO 5000
[DN2] 1704 REM *** NAME ***
[BP3] 1705 IF EIN<>132 THEN 1714
[E52] 1706 GOSUB 10030
[S32] 1707 GOSUB 6000
[HK3] 1708 IF TYP<>4 THEN 9980
[462] 1709 GOSUB 12900
[N32] 1710 GOSUB 6000
[EK3] 1711 IF TYP<>4 THEN 9980
[3F3] 1712 UP=&H8E:GOSUB 11200
[NW1] 1713 GOTO 5000
[WR2] 1714 REM *** DATES ***
[HP3] 1715 IF EIN<>141 THEN 1723
[K52] 1716 GOSUB 10030
[X48] 1717 IF EIN<>231 THEN PRINT "'=' erwartet";:GOTO
9990
[M52] 1718 GOSUB 10030
[Y32] 1719 GOSUB 6000
[HK3] 1720 IF TYP<>4 THEN 9980
```

```
[SE3] 1721 UP=&H93:GOSUB 11200
[SW1] 1722 GOTO 5000
[Q73] 1723 REM *** Fehler ***
[UL6] 1724 PRINT "Unbekannter Befehl";:GOTO 9990
[AG2] 2000 REM *** ON ***
[KN3] 2001 IF EIN<>149 THEN 2020
[R42] 2002 GOSUB 10030
[952] 2003 GOSUB 11900
[QA2] 2004 UP=EIN=141
[8D4] 2005 IF UP OR EIN=137 THEN 2007
[DT7] 2006 PRINT "GOTO oder GOSUB erwartet";:GOTO 9990
[1M3] 2007 AUS=&HE8:GOSUB 11000
[2C5] 2008 WORT=ONADR(ONCNT):GOSUB 12000
[Y42] 2009 GOSUB 10030
[VT9] 2010 IF EIN<>14 THEN PRINT "Zeilennummer erwartet
";:GOTO 9980
[U42] 2011 GOSUB 13200
[XC4] 2012 WORT=FNI(RCI+FNR!(WORT))
[252] 2013 GOSUB 11040
[NC3] 2014 IF EIN=44 THEN 2009
[AX3] 2015 ONADR(ONCNT)=FNI(PCI)
[7LE] 2016 ONCNT=ONCNT+1:IF ONCNT>499 THEN PRINT "Zu vi
ele ON/GOTO/GOSUB-Anweisungen";:GOTO 9990
[XB2] 2017 UP=&H4E-UP
[X42] 2018 GOSUB 11200
[8W1] 2019 GOTO 5000
[HS2] 2020 REM *** STOP ***
[OP3] 2021 IF EIN<>144 THEN 2033
[Z42] 2022 GOSUB 10030
[SE3] 2023 WORT=0:GOSUB 11080
[AL4] 2024 VAR$="Break in":GOSUB 13100
[BD3] 2025 UP=&H15:GOSUB 11200
[XU5] 2026 WORT=ZEILNR(ZEILE-1):GOSUB 11080
[ED3] 2027 UP=&H16:GOSUB 11200
[ED3] 2028 UP=&H15:GOSUB 11200
[ED3] 2029 UP=&H14:GOSUB 11200
[PL3] 2030 AUS=&H58:GOSUB 11000
[SB3] 2031 UP=&H1:GOSUB 11200
[9W1] 2032 GOTO 5000
[322] 2033 REM *** WRITE ***
[3R3] 2034 IF EIN<>183 THEN 2056
[652] 2035 GOSUB 10030
[MB7] 2036 IF EIN<>35 THEN WORT=0:GOSUB 11080:GOTO 2040
[852] 2037 GOSUB 10030
[U52] 2038 GOSUB 11900
[D23] 2039 X$=",";GOSUB 11800
[PS4] 2040 IF EIN=0 OR EIN=58 THEN 2050
[F32] 2041 GOSUB 6000
[225] 2042 ON TYP GOTO 2043,2044,9980,2047
[753] 2043 UP=&H16:GOTO 2045
[YP1] 2044 UP=&H46
[652] 2045 GOSUB 11200
[763] 2046 UP=&H72:GOTO 2048
[AR1] 2047 UP=&H73
[952] 2048 GOSUB 11200
[SE5] 2049 IF EIN<>44 AND EIN<>59 THEN 2053
[952] 2050 GOSUB 10030
[AE3] 2051 UP=&H74:GOSUB 11200
[GW1] 2052 GOTO 2041
[MD3] 2053 UP=&H14:GOSUB 11200
[2M3] 2054 AUS=&H58:GOSUB 11000
[MW1] 2055 GOTO 5000
[5R2] 2056 REM *** WAIT ***
[4R3] 2057 IF EIN<>150 THEN 2067
[H52] 2058 GOSUB 10030
[362] 2059 GOSUB 11900
[G23] 2060 X$=",";GOSUB 11800
[Z52] 2061 GOSUB 11900
[7C7] 2062 IF EIN<>44 THEN WORT=0:GOSUB 11080:GOTO 2065
[G52] 2063 GOSUB 10030
[262] 2064 GOSUB 11900
[LE3] 2065 UP=&H75:GOSUB 11200
[TW1] 2066 GOTO 5000
[6T2] 2067 REM *** SWAP ***
[3R3] 2068 IF EIN<>164 THEN 2081
[P52] 2069 GOSUB 10030
[162] 2070 GOSUB 10070
[HU2] 2071 SWAPTYP=TYP
[P23] 2072 X$=",";GOSUB 11800
[462] 2073 GOSUB 10070
[2X4] 2074 IF TYP<>SWAPTYP THEN 9980
[P35] 2075 ON TYP GOTO 2076,2077,9980,2078
[173] 2076 UP=&H77:GOTO 2079
[373] 2077 UP=&H78:GOTO 2079
[XR1] 2078 UP=&H79
[P52] 2079 GOSUB 11200
[VW1] 2080 GOTO 5000
[UY2] 2081 REM *** SOUND ***
[BS3] 2082 IF EIN<>196 THEN 2089
[S52] 2083 GOSUB 10030
[Y62] 2084 GOSUB 11950
[X23] 2085 X$=",";GOSUB 11800
[W72] 2086 GOSUB 11950
[9F3] 2087 UP=&H7A:GOSUB 11200
[3X1] 2088 GOTO 5000
[P73] 2089 REM *** SCREEN ***
[ZN3] 2090 IF EIN<>200 THEN 2101
[U52] 2091 GOSUB 10030
[C62] 2092 GOSUB 11900
```



```
[TC7] 2093 IF EIN<>44 THEN WORT=0:GOSUB 11080:GOTO 2096
[X52] 2094 GOSUB 10030
[F62] 2095 GOSUB 11900
[2D7] 2096 IF EIN<>44 THEN WORT=0:GOSUB 11080:GOTO 2099
[062] 2097 GOSUB 10030
[K62] 2098 GOSUB 11900
[KF3] 2099 UP=&H7D:GOSUB 11200
[YV1] 2100 GOTO 5000
[1W2] 2101 REM *** CLEAR ***
[8P3] 2102 IF EIN<>146 THEN 2108
[V42] 2103 GOSUB 10030
[KM3] 2104 AUS=&HBB:GOSUB 11000
[EF6] 2105 WORT=FNI(INT((VC1+15)/16))+SC:GOSUB 11040
[FE3] 2106 UP=&H7E:GOSUB 11200
[5W1] 2107 GOTO 5000
[YL2] 2108 REM *** RUN ***
[KP3] 2109 IF EIN<>138 THEN 2119
[W42] 2110 GOSUB 10030
[LM3] 2111 AUS=&HBB:GOSUB 11000
[516] 2112 WORT=FNI(INT((VC1+15)/16)):GOSUB 11040
[GE3] 2113 UP=&H7E:GOSUB 11200
[WAB] 2114 IF EIN=0 OR EIN=58 THEN WORT=ZEILADR(0):GOTO
2116
[152] 2115 GOSUB 13200
[9M3] 2116 AUS=&HE9:GOSUB 11000
[V42] 2117 GOSUB 12000
[AW1] 2118 GOTO 5000
[MM3] 2119 REM *** RANDOMIZE ***
[SP3] 2120 IF EIN<>185 THEN 2125
[152] 2121 GOSUB 10030
[762] 2122 GOSUB 11950
[5E3] 2123 UP=&H82:GOSUB 11200
[AW1] 2124 GOTO 5000
[0N2] 2125 REM *** BEEP ***
[RP3] 2126 IF EIN<>197 THEN 2130
[752] 2127 GOSUB 10030
[BE3] 2128 UP=&H83:GOSUB 11200
[FW1] 2129 GOTO 5000
[4X2] 2130 REM *** BSAVE ***
[SP3] 2131 IF EIN<>194 THEN 2141
[652] 2132 GOSUB 10030
[G32] 2133 GOSUB 6000
[9K3] 2134 IF TYP<>4 THEN 9980
[C23] 2135 X$="," :GOSUB 11800
[V52] 2136 GOSUB 11900
[E23] 2137 X$="," :GOSUB 11800
[X52] 2138 GOSUB 11900
[KE3] 2139 UP=&H85:GOSUB 11200
[EW1] 2140 GOTO 5000
[EV2] 2141 REM *** BLOAD ***
[0R3] 2142 IF EIN<>195 THEN 2151
[B52] 2143 GOSUB 10030
[N32] 2144 GOSUB 6000
[EK3] 2145 IF TYP<>4 THEN 9980
[8G7] 2146 IF EIN<>44 THEN WORT=-1:GOSUB 11080:GOTO 214
9
[F52] 2147 GOSUB 10030
[162] 2148 GOSUB 11900
[RE3] 2149 UP=&H86:GOSUB 11200
[KW1] 2150 GOTO 5000
[8M2] 2151 REM *** OUT ***
[DR3] 2152 IF EIN<>156 THEN 2159
[F52] 2153 GOSUB 10030
[162] 2154 GOSUB 11900
[M23] 2155 X$="," :GOSUB 11800
[362] 2156 GOSUB 11900
[VE3] 2157 UP=&H88:GOSUB 11200
[UW1] 2158 GOTO 5000
[273] 2159 REM *** Fehler ***
[0B6] 2160 PRINT "Unbekannter Befehl";:GOTO 9990
[M03] 5000 IF EIN THEN 1039
[9Y2] 5001 FOR I=0 TO IP-1
[8H4] 5002 IFADR(IFSTACK(I))=FNI(PC!))
[B91] 5003 NEXT
[671] 5004 IP=0
[FW1] 5005 GOTO 1027
[4M4] 6000 REM *** Ausdruck lesen ***
[652] 6001 GOSUB 10050
[KE9] 6002 IF EIN=0 OR EIN=58 OR EIN=44 OR EIN=59 THEN
TYP=0:RETURN
[VV3] 6003 STACK(SP)=248:SP=SP+1
[204] 6004 REM *** VORZEICHEN ***
[EB9] 6005 IF EIN=234 THEN STACK(SP)=1:SP=SP+1:GOSUB 10
030:GOTO 6004
[P36] 6006 IF EIN=233 THEN GOSUB 10030:GOTO 6004
[4L2] 6007 REM *** NOT ***
[YA9] 6008 IF EIN=211 THEN STACK(SP)=0:SP=SP+1:GOSUB 10
030:GOTO 6004
[U24] 6009 REM *** Integer-Zahl ***
[DF1] 6010 TYP=1
[EU4] 6011 IF EIN<17 OR EIN>26 THEN 6015
[U44] 6012 WORT=EIN-17:GOSUB 11080
[452] 6013 GOSUB 10030
[LW1] 6014 GOTO 7000
[BK3] 6015 IF EIN<>15 THEN 6019
[VE5] 6016 GOSUB 9995:WORT=EIN:GOSUB 11080
[852] 6017 GOSUB 10030
[RW1] 6018 GOTO 7000
[797] 6019 IF EIN<>28 AND EIN<>11 AND EIN<>12 THEN 6023
```

```
[MW3] 6020 GOSUB 10000:GOSUB 11080
[652] 6021 GOSUB 10030
[NW1] 6022 GOTO 7000
[M25] 6023 REM *** Fließkommazahl ***
[LK3] 6024 IF EIN<>29 THEN 6031
[TF1] 6025 TYP=2
[U4] 6026 GOSUB 10000:STACK(SP)=WORT
[VW3] 6027 GOSUB 10000:GOSUB 11080
[YW4] 6028 WORT=STACK(SP):GOSUB 11080
[E52] 6029 GOSUB 10030
[RW1] 6030 GOTO 7000
[CF5] 6031 REM *** Doppelte Genauigkeit ***
[6K3] 6032 IF EIN<>31 THEN 6038
[YF1] 6033 TYP=3
[578] 6034 FOR I=0 TO 3:GOSUB 10000:STACK(SP+I)=WORT:NE
XT
[YA9] 6035 FOR I=3 TO 0 STEP -1:WORT=STACK(SP+I):GOSUB
11080:NEXT
[F52] 6036 GOSUB 10030
[YW1] 6037 GOTO 7000
[YW4] 6038 REM *** Stringkonstante ***
[SK3] 6039 IF EIN<>34 THEN 6047
[2G1] 6040 TYP=4
[YH1] 6041 VAR$=""
[W52] 6042 GOSUB 9995
[959] 6043 IF EIN<>34 AND EIN<>0 THEN VAR$=VAR$+CHR$(EI
N):GOTO 6042
[7L4] 6044 IF EIN=34 THEN GOSUB 10030
[F52] 6045 GOSUB 13100
[1X1] 6046 GOTO 7000
[VB3] 6047 REM *** Variable ***
[TV4] 6048 IF EIN<65 OR EIN>90 THEN 6061
[LE5] 6049 GOSUB 10070:IF CMDFLAG THEN 6600
[W35] 6050 ON TYP GOTO 6054,6057,9980,6051
[AF3] 6051 WORT=0:GOSUB 11080
[TD3] 6052 UP=&H12:GOSUB 11200
[2X1] 6053 GOTO 7000
[YM3] 6054 AUS=&H5B:GOSUB 11000
[B34] 6055 WORT=&H37FF:GOSUB 11040
[5X1] 6056 GOTO 7000
[S34] 6057 WORT=&HFF5B:GOSUB 11040
[AZ3] 6058 WORT=&H277:GOSUB 11040
[F34] 6059 WORT=&H37FF:GOSUB 11040
[3X1] 6060 GOTO 7000
[EF5] 6061 REM *** Ausdruck in Klammern ***
[KK3] 6062 IF EIN<>40 THEN 6067
[S52] 6063 GOSUB 10030
[242] 6064 GOSUB 6000
[M23] 6065 X$="," :GOSUB 11800
[9X1] 6066 GOTO 7000
[814] 6067 REM *** Funktionen ***
[BR3] 6068 IF EIN<>255 THEN 6300
[B62] 6069 GOSUB 9995
[3Y1] 6070 FEIN=EIN
[U52] 6071 GOSUB 10030
[UL2] 6072 REM *** CHR$ ***
[0X3] 6073 IF FEIN<>150 THEN 6080
[M23] 6074 X$="," :GOSUB 11800
[F62] 6075 GOSUB 11900
[CE3] 6076 UP=&H19:GOSUB 11200
[NG1] 6077 TYP=4
[V23] 6078 X$="," :GOSUB 11800
[GK1] 6079 GOTO 7000
[9K2] 6080 REM *** LEN ***
[DY3] 6081 IF FEIN<>146 THEN 6088
[P23] 6082 X$="," :GOSUB 11800
[942] 6083 GOSUB 6000
[2L3] 6084 IF TYP<>4 THEN 9980
[FE3] 6085 UP=&H25:GOSUB 11200
[FG1] 6086 TYP=1
[MY1] 6087 GOTO 6078
[WK2] 6088 REM *** EOF ***
[8Y3] 6089 IF FEIN<>163 THEN 6095
[S23] 6090 X$="," :GOSUB 11800
[L62] 6091 GOSUB 11900
[KE3] 6092 UP=&H27:GOSUB 11200
[GG1] 6093 TYP=1
[NY1] 6094 GOTO 6078
[1R2] 6095 REM *** PEEK ***
[VW3] 6096 IF FEIN<>151 THEN 6101
[Z23] 6097 X$="," :GOSUB 11800
[U62] 6098 GOSUB 11900
[W33] 6099 UP=&H36:GOSUB 11200
[5X1] 6100 GOTO 6093
[TH2] 6101 REM *** ASC ***
[1X3] 6102 IF FEIN<>149 THEN 6108
[U13] 6103 X$="," :GOSUB 11800
[D32] 6104 GOSUB 6000
[6K3] 6105 IF TYP<>4 THEN 9980
[TD3] 6106 UP=&H37:GOSUB 11200
[CX1] 6107 GOTO 6093
[CT2] 6108 REM *** LEFT$ ***
[9X3] 6109 IF FEIN<>129 THEN 6118
[V13] 6110 X$="," :GOSUB 11800
[E32] 6111 GOSUB 6000
```

Listing 1. Der Quelltext des SBasic-Compilers »sbasic.bas«


```
[7K3] 6112 IF TYP<>4 THEN 9980
[SE7] 6113 IF EIN=41 THEN WORT=255:GOSUB 11080:GOTO 611
6
[B23] 6114 X$="":GOSUB 11800
[U52] 6115 GOSUB 11900
[YD3] 6116 UP=&H38:GOSUB 11200
[XX1] 6117 GOTO 6077
[T33] 6118 REM *** RIGHTS ***
[RW3] 6119 IF FEIN<>130 THEN 6126
[Z13] 6120 X$="":GOSUB 11800
[V05] 6121 GOSUB 6000:IF TYP<>4 THEN 9980
[D23] 6122 X$="":GOSUB 11800
[W52] 6123 GOSUB 11900
[LE3] 6124 UP=&H39:GOSUB 11200
[UX1] 6125 GOTO 6077
[LM2] 6126 REM *** MID$ ***
[6X3] 6127 IF FEIN<>131 THEN 6138
[723] 6128 X$="":GOSUB 11800
[U32] 6129 GOSUB 6000
[DK3] 6130 IF TYP<>4 THEN 9980
[G23] 6131 X$="":GOSUB 11800
[Z52] 6132 GOSUB 11900
[4F7] 6133 IF EIN=41 THEN WORT=255:GOSUB 11080:GOTO 613
6
[L23] 6134 X$="":GOSUB 11800
[Z62] 6135 GOSUB 11900
[FE3] 6136 UP=&H3A:GOSUB 11200
[0Y1] 6137 GOTO 6077
[1L2] 6138 REM *** LOG ***
[HX3] 6139 IF FEIN<>138 THEN 6145
[723] 6140 X$="":GOSUB 11800
[P62] 6141 GOSUB 11950
[NE3] 6142 UP=&H4C:GOSUB 11200
[2G1] 6143 TYP=2
[4Y1] 6144 GOTO 6078
[2L2] 6145 REM *** INT ***
[VW3] 6146 IF FEIN<>133 THEN 6151
[E23] 6147 X$="":GOSUB 11800
[X62] 6148 GOSUB 11950
[DE3] 6149 UP=&H50:GOSUB 11200
[GX1] 6150 GOTO 6143
[MP2] 6151 REM *** STR$ ***
[3X3] 6152 IF FEIN<>147 THEN 6160
[E23] 6153 X$="":GOSUB 11800
[142] 6154 GOSUB 6000
[T55] 6155 ON TYP GOTO 6156,6157,9980,9980
[N63] 6156 UP=&H16:GOTO 6158
[GR1] 6157 UP=&H46
[S52] 6158 GOSUB 11200
[AY1] 6159 GOTO 6077
[GK2] 6160 REM *** VAL ***
[4Y3] 6161 IF FEIN<>148 THEN 6167
[H23] 6162 X$="":GOSUB 11800
[442] 6163 GOSUB 6000
[XK3] 6164 IF TYP<>4 THEN 9980
[TE3] 6165 UP=&H64:GOSUB 11200
[UX1] 6166 GOTO 6143
[RL2] 6167 REM *** RND ***
[HY3] 6168 IF FEIN<>136 THEN 6179
[VK3] 6169 IF EIN<>40 THEN 6174
[W52] 6170 GOSUB 10030
[Z72] 6171 GOSUB 11950
[ZB3] 6172 UP=&H69:GOSUB 11200
[VX1] 6173 GOTO 6143
[F04] 6174 WORT=&H8100:GOSUB 11080
[NN3] 6175 WORT=&H0:GOSUB 11080
[3F3] 6176 UP=&H69:GOSUB 11200
[KG1] 6177 TYP=2
[KX1] 6178 GOTO 7000
[AM2] 6179 REM *** POS ***
[7Y3] 6180 IF FEIN<>145 THEN 6186
[S23] 6181 X$="":GOSUB 11800
[L62] 6182 GOSUB 11900
[SM3] 6183 AUS=&H58:GOSUB 11000
[CF3] 6184 UP=&H5D:GOSUB 11200
[HY1] 6185 GOTO 6086
[S52] 6186 REM *** CINT ***
[4Y3] 6187 IF FEIN<>156 THEN 6192
[Z23] 6188 X$="":GOSUB 11800
[U62] 6189 GOSUB 11900
[GY1] 6190 GOTO 6086
[TK2] 6191 REM *** FIX ***
[XY3] 6192 IF FEIN<>159 THEN 6197
[Y23] 6193 X$="":GOSUB 11800
[D72] 6194 GOSUB 11950
[RF3] 6195 UP=&H6F:GOSUB 11200
[6Y1] 6196 GOTO 6143
[LM2] 6197 REM *** HEX$ ***
[GX3] 6198 IF FEIN<>154 THEN 6203
[433] 6199 X$="":GOSUB 11800
[M52] 6200 GOSUB 11900
[0E3] 6201 UP=&H70:GOSUB 11200
[KX1] 6202 GOTO 6077
[FK2] 6203 REM *** INP ***
[2X3] 6204 IF FEIN<>144 THEN 6209
[Z13] 6205 X$="":GOSUB 11800
[U52] 6206 GOSUB 11900
[7E3] 6207 UP=&H71:GOSUB 11200
[RX1] 6208 GOTO 6086
```

```
[V23] 6209 REM *** SPACES ***
[KW3] 6210 IF FEIN<>152 THEN 6215
[Z13] 6211 X$="":GOSUB 11800
[U52] 6212 GOSUB 11900
[EE3] 6213 UP=&H84:GOSUB 11200
[XX1] 6214 GOTO 6077
[GG2] 6215 REM *** ABS ***
[TW3] 6216 IF FEIN<>134 THEN 6224
[S23] 6217 X$="":GOSUB 11800
[S32] 6218 GOSUB 6000
[E45] 6219 ON TYP GOTO 6220,6221,9980,9980
[P63] 6220 UP=&H8A:GOTO 6222
[XR1] 6221 UP=&H8B
[Y52] 6222 GOSUB 11200
[YX1] 6223 GOTO 6078
[SK2] 6224 REM *** SGN ***
[RW3] 6225 IF FEIN<>132 THEN 6233
[823] 6226 X$="":GOSUB 11800
[V32] 6227 GOSUB 6000
[Z55] 6228 ON TYP GOTO 6229,6230,9980,9980
[363] 6229 UP=&H1A:GOTO 6222
[BR1] 6230 UP=&H59
[C52] 6231 GOSUB 11200
[XX1] 6232 GOTO 6086
[XY2] 6233 REM *** STICK ***
[LX3] 6234 IF FEIN<>161 THEN 6239
[B23] 6235 X$="":GOSUB 11800
[662] 6236 GOSUB 11900
[CF3] 6237 UP=&H8F:GOSUB 11200
[3Y1] 6238 GOTO 6086
[C22] 6239 REM *** STRIG ***
[CK3] 6240 IF FEIN<>162 THEN 6246
[B23] 6241 X$="":GOSUB 11800
[662] 6242 GOSUB 11900
[UE3] 6243 UP=&H90:GOSUB 11200
[3Y1] 6244 GOTO 6086
[XL2] 6245 REM *** EXP ***
[CK3] 6246 IF FEIN<>139 THEN 6251
[H23] 6247 X$="":GOSUB 11800
[072] 6248 GOSUB 11950
[1F3] 6249 UP=&H91:GOSUB 11200
[LX1] 6250 GOTO 6143
[DM2] 6251 REM *** SQR ***
[WK3] 6252 IF FEIN<>135 THEN 6257
[H23] 6253 X$="":GOSUB 11800
[072] 6254 GOSUB 11950
[2F3] 6255 UP=&H92:GOSUB 11200
[TX1] 6256 GOTO 6143
[FL2] 6257 REM *** SIN ***
[TK3] 6258 IF FEIN<>137 THEN 6263
[R23] 6259 X$="":GOSUB 11800
[072] 6260 GOSUB 11950
[5F3] 6261 UP=&H95:GOSUB 11200
[TK1] 6262 GOTO 6143
[WK2] 6263 REM *** COS ***
[4Y3] 6264 IF FEIN<>140 THEN 6269
[R23] 6265 X$="":GOSUB 11800
[672] 6266 GOSUB 11950
[CF3] 6267 UP=&H96:GOSUB 11200
[ZX1] 6268 GOTO 6143
[SK2] 6269 REM *** TAN ***
[TK3] 6270 IF FEIN<>141 THEN 6275
[R23] 6271 X$="":GOSUB 11800
[672] 6272 GOSUB 11950
[DF3] 6273 UP=&H97:GOSUB 11200
[ZX1] 6274 GOTO 6143
[BL2] 6275 REM *** ATN ***
[MX3] 6276 IF FEIN<>142 THEN 6281
[X23] 6277 X$="":GOSUB 11800
[C72] 6278 GOSUB 11950
[MF3] 6279 UP=&H98:GOSUB 11200
[ZX1] 6280 GOTO 6143
[H73] 6281 REM *** Fehler ***
[LX6] 6282 PRINT "Unbekannte Funktion";:GOTO 9990
[T33] 6300 REM *** INKEY$ ***
[SP3] 6301 IF EIN<>222 THEN 6306
[852] 6302 GOSUB 10030
[TD3] 6303 UP=&H28:GOSUB 11200
[ZF1] 6304 TYP=4
[TW1] 6305 GOTO 7000
[N43] 6306 REM *** INPUT$ ***
[8R3] 6307 IF EIN<>133 THEN 6318
[E52] 6308 GOSUB 10030
[U13] 6309 X$="":GOSUB 11800
[Z23] 6310 X$="":GOSUB 11800
[W52] 6311 GOSUB 11900
[TC7] 6312 IF EIN<>44 THEN WORT=0:GOSUB 11080:GOTO 6316
[D52] 6313 GOSUB 10030
[7L4] 6314 IF EIN=35 THEN GOSUB 10030
[062] 6315 GOSUB 11900
[EE3] 6316 UP=&H2F:GOSUB 11200
[YX1] 6317 GOTO 6077
[8Z2] 6318 REM *** INSTR ***
[DR3] 6319 IF EIN<>216 THEN 6337
[E52] 6320 GOSUB 10030
[BF3] 6321 WORT=1:GOSUB 11080
[Z23] 6322 X$="":GOSUB 11800
[U32] 6323 GOSUB 6000
[ZV5] 6324 INFLAG=TYP=4:IF INFLAG THEN 6329
```



```
[AV3] 6325 NEUTYP=1:GOSUB 11600
[R23] 6326 X$=",":GOSUB 11800
[Y32] 6327 GOSUB 6000
[RK3] 6328 IF TYP<>4 THEN 9980
[U23] 6329 X$=",":GOSUB 11800
[V32] 6330 GOSUB 6000
[MK3] 6331 IF TYP<>4 THEN 9980
[1E3] 6332 UP=&H31:GOSUB 11200
[BH3] 6333 IF INFLAG THEN 6086
[Y14] 6334 WORT=&H5B58:GOSUB 11040
[WL3] 6335 AUS=&H50:GOSUB 11000
[4Y1] 6336 GOTO 6086
[LG2] 6337 REM *** FN ***
[LR3] 6338 IF EIN<>209 THEN 6365
[V52] 6339 GOSUB 10030
[S52] 6340 GOSUB 11400
[521] 6341 I=0
[41A] 6342 IF I=FUNNR THEN PRINT "Undefinierte Funktion
";:GOTO 9990
[FF7] 6343 IF VARNAME$(I) THEN I=I+1:GOTO 634
2
[FK7] 6344 STACK(SP)=I:SP=SP+1:STACK(SP)=0:SP=SP+1:J=3
[V74] 6345 STACK(SP)=FUNPART(I,J)
[YK4] 6346 IF STACK(SP) THEN SP=SP+1
[RV3] 6347 J=J-1:IF J=0 THEN 6345
[6L3] 6348 IF EIN<>40 THEN 6358
[Z52] 6349 GOSUB 10030
[342] 6350 GOSUB 6000
[RS4] 6351 SP=SP-1:NEUTYP=STACK(SP)
[5R9] 6352 IF NEUTYP=0 THEN PRINT "Zu viele Parameter";
:GOTO 9990
[562] 6353 GOSUB 11600
[GE3] 6354 IF EIN=41 THEN 6357
[E39] 6355 IF EIN<>44 THEN PRINT "',' oder ')' erwartet
";:GOTO 9990
[GY1] 6356 GOTO 6349
[162] 6357 GOSUB 10030
[2U1] 6358 SP=SP-1
[KP9] 6359 IF STACK(SP) THEN PRINT "Zu wenige Parameter
";:GOTO 9990
[8U3] 6360 SP=SP-1:I=STACK(SP)
[4N3] 6361 AUS=&HE8:GOSUB 11000
[UT4] 6362 WORT=FUNANF(I):GOSUB 12000
[PO3] 6363 TYP=FUNTYP(I)
[GX1] 6364 GOTO 7000
[QB3] 6365 REM *** STRING$ ***
[4S3] 6366 IF EIN<>214 THEN 6377
[562] 6367 GOSUB 10030
[X23] 6368 X$="(:GOSUB 11800
[S62] 6369 GOSUB 11900
[533] 6370 X$=",":GOSUB 11800
[C42] 6371 GOSUB 6000
[D55] 6372 ON TYP GOTO 6374,6373,9980,6375
[UE3] 6373 UP=&H51:GOSUB 11200
[LE3] 6374 UP=&H19:GOSUB 11200
[5F3] 6375 UP=&H3B:GOSUB 11200
[NY1] 6376 GOTO 6077
[NA3] 6377 REM *** VARPTR ***
[4S3] 6378 IF EIN<>218 THEN 6383
[B62] 6379 GOSUB 10030
[X23] 6380 X$="(:GOSUB 11800
[R62] 6381 GOSUB 12800
[MY1] 6382 GOTO 6086
[YU2] 6383 REM *** TIMES ***
[VR3] 6384 IF EIN<>254 THEN 6700
[UM3] 6385 GOSUB 9995:FEIN=EIN
[C62] 6386 GOSUB 10030
[1Y3] 6387 IF FEIN<>142 THEN 6391
[GF3] 6388 UP=&H3F:GOSUB 11200
[4H1] 6389 TYP=4
[SX1] 6390 GOTO 7000
[RS2] 6391 REM *** DATE$ ***
[EY3] 6392 IF FEIN<>141 THEN 6395
[RF3] 6393 UP=&H6B:GOSUB 11200
[6Z1] 6394 GOTO 6389
[EZ2] 6395 REM *** TIMER ***
[LX3] 6396 IF FEIN<>148 THEN 6400
[WF3] 6397 UP=&H6C:GOSUB 11200
[1H1] 6398 TYP=2
[1Y1] 6399 GOTO 7000
[P63] 6400 REM *** Fehler ***
[SW6] 6401 PRINT "Unbekannte Funktion";:GOTO 9990
[YA3] 6600 REM *** COMMAND$ ***
[723] 6601 X$="(:GOSUB 11800
[262] 6602 GOSUB 11900
[UE3] 6603 UP=&H94:GOSUB 11200
[0Y1] 6604 GOTO 6077
[KM2] 6700 REM *** MKI$ ***
[4R3] 6701 IF EIN<>253 THEN 6800
[3M3] 6702 GOSUB 9995:FEIN=EIN
[N52] 6703 GOSUB 10030
[PX3] 6704 IF FEIN<>132 THEN 6709
[E23] 6705 X$="(:GOSUB 11800
[962] 6706 GOSUB 11900
[SE3] 6707 UP=&H67:GOSUB 11200
[7Y1] 6708 GOTO 6077
[5N2] 6709 REM *** MKS$ ***
[BX3] 6710 IF FEIN<>133 THEN 6715
[E23] 6711 X$="(:GOSUB 11800
```

```
[X62] 6712 GOSUB 11950
[TE3] 6713 UP=&H68:GOSUB 11200
[7Y1] 6714 GOTO 6077
[BL2] 6715 REM *** CVI ***
[LX3] 6716 IF FEIN<>129 THEN 6722
[M23] 6717 X$="(:GOSUB 11800
[742] 6718 GOSUB 6000
[0L3] 6719 IF TYP<>4 THEN 9980
[6F3] 6720 UP=&H6D:GOSUB 11200
[7Y1] 6721 GOTO 6086
[PL2] 6722 REM *** CVS ***
[ZX3] 6723 IF FEIN<>130 THEN 6729
[N23] 6724 X$="(:GOSUB 11800
[842] 6725 GOSUB 6000
[1L3] 6726 IF TYP<>4 THEN 9980
[EF3] 6727 UP=&H6E:GOSUB 11200
[YX1] 6728 GOTO 6143
[Z21] 6729 REM *** ***
[H93] 6730 PRINT FEIN:STOP
[D73] 6800 REM *** SCREEN ***
[NP3] 6801 IF EIN<>200 THEN 6811
[R52] 6802 GOSUB 10030
[F23] 6803 X$="(:GOSUB 11800
[A62] 6804 GOSUB 11900
[X23] 6805 X$=",":GOSUB 11800
[C62] 6806 GOSUB 11900
[X57] 6807 IF EIN=44 THEN GOSUB 10030:GOSUB 11900:GOTO
6809
[NF3] 6808 WORT=0:GOSUB 11080
[7F3] 6809 UP=&H6A:GOSUB 11200
[5Y1] 6810 GOTO 6086
[W83] 6811 REM *** CSRLIN ***
[TR3] 6812 IF EIN<>219 THEN 6817
[W52] 6813 GOSUB 10030
[4F3] 6814 UP=&H5C:GOSUB 11200
[AG1] 6815 TYP=1
[DX1] 6816 GOTO 7000
[W21] 6817 REM *** ***
[FX6] 6818 PRINT "Unbekannte Funktion";:GOTO 9990
[RF3] 7000 REM *** Operator ***
[834] 7001 TYPEN(TP)=TYP:TP=TP+1
[U71] 7002 OP=0
[AX7] 7003 IF EIN=230 THEN OP=OP OR 41:GOSUB 10030:GOTO
7003
[GX7] 7004 IF EIN=231 THEN OP=OP OR 42:GOSUB 10030:GOTO
7003
[RX7] 7005 IF EIN=232 THEN OP=OP OR 44:GOSUB 10030:GOTO
7003
[VU2] 7006 IF OP THEN 7019
[LV3] 7007 IF EIN=233 THEN OP=32
[VW3] 7008 IF EIN=234 THEN OP=33
[1X3] 7009 IF EIN=235 THEN OP=24
[YW3] 7010 IF EIN=244 THEN OP=25
[YW3] 7011 IF EIN=243 THEN OP=26
[BX3] 7012 IF EIN=236 THEN OP=27
[VX3] 7013 IF EIN=238 THEN OP=48
[3Y3] 7014 IF EIN=239 THEN OP=49
[AW3] 7015 IF EIN=240 THEN OP=50
[EX3] 7016 IF EIN=237 THEN OP=16
[715] 7017 IF OP=0 THEN OP=240:GOTO 7019
[B52] 7018 GOSUB 10030
[V53] 7019 IF SP=0 THEN 7127
[N47] 7020 IF (STACK(SP-1) AND 248)>(OP AND 248) THEN 7
127
[8T1] 7021 SP=SP-1
[MD3] 7022 REM *** Negieren ***
[0E4] 7023 IF STACK(SP)<>1 THEN 7027
[996] 7024 ON TYPEN(TP-1) GOTO 7025,7026,9980,9980
[463] 7025 UP=&H11:GOTO 7039
[R63] 7026 UP=&H47:GOTO 7039
[EL2] 7027 REM *** NOT ***
[RD4] 7028 IF STACK(SP)<>0 THEN 7032
[R86] 7029 ON TYPEN(TP-1) GOTO 7031,7030,9980,9980
[H52] 7030 GOSUB 13400
[363] 7031 UP=&H10:GOTO 7039
[C52] 7032 GOSUB 11300
[VU2] 7033 REM *** »« ***
[WK4] 7034 IF STACK(SP)<>32 THEN 7041
[B86] 7035 ON TYPEN(TP-1) GOTO 7036,7037,9980,7038
[Y23] 7036 UP=&H8:GOTO 7039
[X63] 7037 UP=&H48:GOTO 7039
[YP1] 7038 UP=&H18
[G52] 7039 GOSUB 11200
[UX1] 7040 GOTO 7019
[TU2] 7041 REM *** »« ***
[XL4] 7042 IF STACK(SP)<>24 THEN 7048
[H96] 7043 ON TYPEN(TP-1) GOTO 7044,7045,9980,9980
[G43] 7044 UP=&H4:GOTO 7039
[873] 7045 UP=&H4A:GOTO 7039
[XL2] 7046 TYPEN(TP-1)=1
[5Y1] 7047 GOTO 7039
[8X2] 7048 REM *** »« ***
[TL4] 7049 IF STACK(SP)<>44 THEN 7054
```

Listing 1. Der Quelltext des SBasic-Compilers »sbasic.bas«


```
[Y76] 7050 ON TYPEN(TP-1) GOTO 7051,7052,9980,7053
[823] 7051 UP=&H3:GOTO 7046
[S63] 7052 UP=&H53:GOTO 7046
[V63] 7053 UP=&H2A:GOTO 7046
[CX2] 7054 REM *** »« ***
[8L4] 7055 IF STACK(SP)<>42 THEN 7060
[896] 7056 ON TYPEN(TP-1) GOTO 7057,7058,9980,7059
[K23] 7057 UP=&H4:GOTO 7046
[Z63] 7058 UP=&H54:GOTO 7046
[T63] 7059 UP=&H29:GOTO 7046
[GX2] 7060 REM *** »« ***
[YL4] 7061 IF STACK(SP)<>41 THEN 7066
[R86] 7062 ON TYPEN(TP-1) GOTO 7063,7064,9980,7065
[A23] 7063 UP=&H2:GOTO 7046
[X63] 7064 UP=&H52:GOTO 7046
[Z73] 7065 UP=&H2B:GOTO 7046
[F43] 7066 REM *** »« ***
[LM4] 7067 IF STACK(SP)<>46 THEN 7072
[Z86] 7068 ON TYPEN(TP-1) GOTO 7069,7070,9980,7071
[Q33] 7069 UP=&H6:GOTO 7046
[L73] 7070 UP=&H56:GOTO 7046
[Z73] 7071 UP=&H2C:GOTO 7046
[R43] 7072 REM *** »« ***
[LM4] 7073 IF STACK(SP)<>43 THEN 7078
[G96] 7074 ON TYPEN(TP-1) GOTO 7075,7076,9980,7077
[W23] 7075 UP=&H5:GOTO 7046
[G73] 7076 UP=&H55:GOTO 7046
[A73] 7077 UP=&H2D:GOTO 7046
[P43] 7078 REM *** »« ***
[GM4] 7079 IF STACK(SP)<>45 THEN 7084
[Z86] 7080 ON TYPEN(TP-1) GOTO 7081,7082,9980,7083
[A33] 7081 UP=&H7:GOTO 7046
[B73] 7082 UP=&H57:GOTO 7046
[B73] 7083 UP=&H2E:GOTO 7046
[8K2] 7084 REM *** AND ***
[AN4] 7085 IF STACK(SP)<>48 THEN 7089
[YA6] 7086 ON TYPEN(TP-1) GOTO 7087,7088,9980,9980
[C62] 7087 GOSUB 13400
[S43] 7088 UP=&HA:GOTO 7039
[DK2] 7089 REM *** OR ***
[UM4] 7090 IF STACK(SP)<>49 THEN 7094
[7A6] 7091 ON TYPEN(TP-1) GOTO 7092,9980,9980
[B62] 7092 GOSUB 13400
[V43] 7093 UP=&HB:GOTO 7039
[3N2] 7094 REM *** XOR ***
[ZN4] 7095 IF STACK(SP)<>50 THEN 7099
[5B6] 7096 ON TYPEN(TP-1) GOTO 7097,7098,9980,9980
[G62] 7097 GOSUB 13400
[453] 7098 UP=&HC:GOTO 7039
[X13] 7099 REM *** »« ***
[PK4] 7100 IF STACK(SP)<>25 THEN 7104
[G86] 7101 ON TYPEN(TP-1) GOTO 7102,7103,9980,9980
[A52] 7102 GOSUB 13400
[643] 7103 UP=&HE:GOTO 7039
[LK2] 7104 REM *** MOD ***
[HL4] 7105 IF STACK(SP)<>26 THEN 7109
[E96] 7106 ON TYPEN(TP-1) GOTO 7107,7108,9980,9980
[F52] 7107 GOSUB 13400
[F43] 7108 UP=&HF:GOTO 7039
[QV2] 7109 REM *** »« ***
[YK4] 7110 IF STACK(SP)<>33 THEN 7115
[S86] 7111 ON TYPEN(TP-1) GOTO 7112,7113,9980,9980
[T23] 7112 UP=&H9:GOTO 7039
[QE3] 7113 UP=&H47:GOSUB 11200
[N63] 7114 UP=&H48:GOTO 7039
[8V2] 7115 REM *** »« ***
[ZL4] 7116 IF STACK(SP)<>27 THEN 7122
[G96] 7117 ON TYPEN(TP-1) GOTO 7118,7121,9980,9980
[3E3] 7118 UP=&H45:GOSUB 11200
[8E3] 7119 UP=&H49:GOSUB 11200
[NL2] 7120 TYPEN(TP-1)=2
[Q73] 7121 UP=&H4B:GOTO 7039
[S13] 7122 REM *** »« ***
[HW4] 7123 IF STACK(SP)<>16 THEN STOP
[H96] 7124 ON TYPEN(TP-1) GOTO 7125,7126,9980,9980
[KL8] 7125 UP=&H45:GOSUB 11200:UP=&H49:GOSUB 11200:TYPE
N(TP-1)=2
[773] 7126 UP=&H4D:GOTO 7039
[R39] 7127 IF OP=240 THEN SP=SP-1:TP=TP-1:TYP=TYPEN(TP)
:RETURN
[DX3] 7128 STACK(SP)=OP:SP=SP+1
[9X1] 7129 GOTO 6004
[EW4] 7130 IF STACK(SP)<>33 THEN STOP
[G96] 7131 ON TYPEN(TP-1) GOTO 7132,7133,9980,9980
[K23] 7132 UP=&H9:GOTO 7041
[8E3] 7133 UP=&H47:GOSUB 11200
[763] 7134 UP=&H48:GOTO 7041
[T39] 7135 IF OP=240 THEN SP=SP-1:TP=TP-1:TYP=TYPEN(TP)
:RETURN
[FX3] 7136 STACK(SP)=OP:SP=SP+1
[BX1] 7137 GOTO 6004
[YP4] 9980 REM *** Typ Mischmasch ***
[BD7] 9981 PRINT "Unverträgliche Variablentypen";
[SS5] 9990 REM *** Zeilennummer ausgeben ***
[906] 9991 PRINT " in Zeile";ZEILNR(ZEILE-1)
[GH1] 9992 CLOSE
[Z51] 9993 END
```

```
[LY3] 9995 REM *** Byte lesen ***
[BP6] 9996 IF EINZ>128 THEN GET #1,I1:I1=I1+1:EINZ=1
[PP4] 9997 EIN=ASC(MID$(EINZ,EINZ,1))
[9G2] 9998 EINZ=EINZ+1
[GX1] 9999 RETURN
[VT3] 10000 REM *** Wort lesen ***
[HV3] 10001 GOSUB 9995:WORT=EIN
[N16] 10002 GOSUB 9995:WORT=FNI(WORT+EIN*256!)
[KV1] 10003 RETURN
[P76] 10030 REM *** Leerzeichen überlesen ***
[N82] 10031 GOSUB 9995
[BH3] 10050 IF EIN=32 THEN 10031
[LV1] 10051 RETURN
[SD5] 10070 REM *** Variablenname lesen ***
[S92] 10071 GOSUB 11400
[6BD] 10072 VARDIM=EIN=40:CMDFLAG=(VARNAMES="COMMAND$")
AND VARDIM:IF CMDFLAG THEN RETURN
[Q68] 10073 IF VARDIM THEN VARNAMES$=VARNAMES$+"(":"GOSUB
10030
[UU3] 10074 IF FUNCNT THEN 10107
[U94] 10075 IF VARIABLE=0 THEN 10078
[P31] 10076 I=0
[T4B] 10077 IF VARNAMES$=VARNAMES$(I) THEN 10085 ELSE I=I
+1:IF I<VARIABLE THEN 10077
[9W3] 10078 REM *** Neu anlegen ***
[TE2] 10079 I=VARIABLE
[4AE] 10080 VARIABLE=VARIABLE+1:IF VARIABLE>MAXVAR THEN
PRINT "Zu viele Variablen";:GOTO 9990
[UV3] 10081 VARNAMES$(I)=VARNAMES$
[Z9A] 10082 IF VARDIM THEN PRINT "Feld nicht dimensioniert";:GOTO 9990
[YMK] 10083 IF TYP<4 THEN VARADR(I)=FNI(VCI):VCI=VCI+2`
TYP:IF VCI<65536! THEN 10085 ELSE PRINT "Variablen
speicher voll";:GOTO 9990
[R24] 10084 VARADR(I)=SC:SC=SC+16
[127] 10085 REM *** Adresse der Variable auf Stapel ***
[7T8] 10086 IF TYP<4 THEN WORT=VARADR(I):GOSUB 11080:GO
TO 10089
[ZT4] 10087 ADR=VARADR(I):STRADR=ADR
[PA2] 10088 GOSUB 12700
[SF4] 10089 IF VARDIM=0 THEN RETURN
[YP4] 10090 REM *** Index berechnen ***
[GC4] 10091 TYPEN(TP)=TYP:TP=TP+1
[FU3] 10092 STACK(SP)=0:SP=SP+1
[6S5] 10093 FOR WORT=VARDIM(I,0) TO 1 STEP -1
[BZ5] 10094 STACK(SP)=VARDIM(I,WORT):SP=SP+1
[YE1] 10095 NEXT
[9K4] 10096 SP=SP-1:WORT=STACK(SP)
[LBA] 10097 IF WORT=0 THEN PRINT "Zu viele Indizes ange
geben";:GOTO 9990
[QAZ] 10098 GOSUB 11080
[YA2] 10099 GOSUB 11900
[VB3] 10100 UP=&HD:GOSUB 11200
[HB3] 10101 UP=&H8:GOSUB 11200
[W66] 10102 IF EIN=44 THEN GOSUB 10030:GOTO 10096
[963] 10103 X$="":GOSUB 11800
[5FC] 10104 SP=SP-1:IF STACK(SP)<>0 THEN PRINT "Zu weni
g Indizes angegeben";:GOTO 9990
[RF4] 10105 TP=TP-1:TYP=TYPEN(TP)
[R22] 10106 GOTO 10050
[XC6] 10107 REM *** Lokale Funktionsvariable ***
[R31] 10108 I=0
[ZT5] 10109 IF VARNAMES$=FUNPAR$(I) THEN 10113
[ZE1] 10110 I=I+1
[304] 10111 IF I<FUNCNT THEN 10109
[X22] 10112 GOTO 10075
[GM6] 10113 IF TYP=4 THEN ADR=FUNADR(I):GOTO 12700
[DS4] 10114 WORT=FUNADR(I):GOTO 11080
[MB4] 11000 REM *** Byte ausgeben ***
[D12] 11001 PC1=PC1+1
[F94] 11002 IF PASS=0 THEN RETURN
[WN6] 11003 IF AUSZ>254 THEN PRINT #2,AUSZ$:AUSZ=0
[7M2] 11004 AUSZ=AUSZ+1
[9T4] 11005 MID$(AUSZ$,AUSZ,1)=CHR$(AUS)
[VV1] 11006 RETURN
[BF4] 11040 REM *** Wort ausgeben ***
[N12] 11041 PC1=PC1+2
[L94] 11042 IF PASS=0 THEN RETURN
[GF8] 11043 IF AUSZ>253 THEN PRINT #2,LEFT$(AUSZ$,
:AUSZ=0
[DM2] 11044 AUSZ=AUSZ+2
[475] 11045 MID$(AUSZ$,AUSZ-1,2)=MKI$(WORT)
[ZV1] 11046 RETURN
[3H4] 11080 REM *** Wort auf Stapel ***
[122] 11081 PC1=PC1+4
[R94] 11082 IF PASS=0 THEN RETURN
[HM3] 11083 AUS=&HB8:GOSUB 11003
[W92] 11084 GOSUB 11043
[RD3] 11085 AUS=&H50:GOTO 11003
[Q46] 11200 REM *** Unterprogramm aufrufen ***
[KA3] 11201 IF PASS THEN 11207
[BF4] 11202 IF UPADR(UP)>0 THEN 11205
[KF2] 11203 UPADR(UP)=1
[GH3] 11204 RC1=RC1+UPLEN(UP)
[522] 11205 PC1=PC1+3
[3W1] 11206 RETURN
[PM3] 11207 AUS=&HE8:GOSUB 11000
```



```
[8W3] 11208 X1=UPADR(UP)-RC1-PC1-2
[W64] 11209 IF X1<0 THEN X1=X1+65536!
[D64] 11210 WORT=FNI(X1):GOTO 11040
[BX4] 11300 REM *** Typen angleichen ***
[SW1] 11301 TP=TP-1
[9L6] 11302 IF TYPEN(TP-1)=TYPEN(TP) THEN RETURN
[C59] 11303 IF TYPEN(TP-1)=2 AND TYPEN(TP)=1 THEN UP=&H
45:GOTO 11200
[PAB] 11304 IF TYPEN(TP-1)=1 AND TYPEN(TP)=2 THEN TYPEN
(TP-1)=2:UP=&H49:GOTO 11200
[CC8] 11305 PRINT "Unerlaubte Typverknüpfung";:GOTO 999
0
[4E5] 11400 REM *** Variablenname lesen ***
[VAB] 11401 IF EIN<65 OR EIN>90 THEN PRINT "Variablenname
erwartet";:GOTO 9990
[1C2] 11402 VARNAME$=""
[N15] 11403 VARNAME$=VARNAME$+CHR$(EIN)
[B92] 11404 GOSUB 9995
[HEA] 11405 IF (EIN)=65 AND EIN<=90 OR (EIN)=48 AND EI
N<=57) OR EIN=46 THEN 11403
[0A2] 11406 GOSUB 10050
[8F6] 11407 REM *** Typ der Variable bestimmen ***
[UM4] 11408 TYP=INSTR("&I$","CHR$(EIN)")
[VX5] 11409 IF TYP THEN GOSUB 9995:GOTO 11412
[T65] 11410 TYP=TYPDEF(ASC(VARNAME$)-65)
[VT3] 11411 IF TYP=0 THEN TYP=2
[X25] 11412 VARNAME$=VARNAME$+MID$("&I$","TYP,1)
[132] 11413 GOTO 10050
[FN4] 11600 REM *** Typumwandlung ***
[HA5] 11601 IF TYP=NEUTYP THEN RETURN
[ZK8] 11602 IF TYP=1 AND NEUTYP=2 THEN TYP=2:UP=&H45:GO
TO 11200
[LK8] 11603 IF TYP=2 AND NEUTYP=1 THEN TYP=1:UP=&H51:GO
TO 11200
[LF1] 11604 STOP
[A85] 11800 REM *** Auf Zeichen prüfen ***
[ZN8] 11801 IF EIN<ASC(X$) THEN PRINT X$;" erwartet";:
GOTO 9990
[632] 11802 GOTO 10030
[PU5] 11900 REM *** Integer-Ausdruck holen ***
[F72] 11901 GOSUB 6000
[234] 11902 IF TYP=0 THEN RETURN
[NT3] 11903 NEUTYP=1:GOTO 11600
[DY5] 11950 REM *** Single-Ausdruck holen ***
[M72] 11951 GOSUB 6000
[VT3] 11952 NEUTYP=2:GOTO 11600
[ME6] 12000 REM *** Relative Adresse schreiben ***
[NK3] 12001 X1=FNRI(WORT)-PC1-2
[RP4] 12002 IF X1<-32767 THEN X1=X1+65536!
[ZK2] 12003 WORT=FNI(X1)
[L22] 12004 GOTO 11040
[6R6] 12100 REM *** Variable für FOR-NEXT lesen ***
[V92] 12101 GOSUB 11400
[NK9] 12102 IF TYP=4 THEN PRINT "String als Laufvariabl
e";:GOTO 9990
[1X9] 12103 IF EIN=40 THEN PRINT "Indizierte Laufvariabl
e";:GOTO 9990
[N31] 12104 I=0
[MD4] 12105 IF I=VARIABLE THEN 12108
[BP7] 12106 IF VARNAME$<>VARNAME$(I) THEN I=I+1:GOTO 12
105
[S22] 12107 GOTO 12112
[AW3] 12108 VARNAME$(I)=VARNAME$
[GG3] 12109 VARADR(I)=FNI(VCI)
[5X2] 12110 VCI=VCI+2*TYP
[4X3] 12111 VARIABLE=VARIABLE+1
[5W2] 12112 ADI=VARADR(I)
[XV1] 12113 RETURN
[857] 12300 REM *** Variable FOR-NEXT/FN von Stapel ***
[1N4] 12301 FOR I=0 TO 2*TYP-2 STEP 2
[034] 12302 WORT=&H68F:GOSUB 11040
[S44] 12303 WORT=ADI+I:GOSUB 11040
[5F1] 12304 NEXT
[8W1] 12305 RETURN
[V58] 12400 REM *** Variable FOR-NEXT/FN/CALL auf Stape
l ***
[NN4] 12401 FOR I=2*TYP-2 TO 0 STEP -2
[394] 12402 WORT=&H36FF:GOSUB 11040
[W44] 12403 WORT=ADI+I:GOSUB 11040
[9F1] 12404 NEXT
[CW1] 12405 RETURN
[R24] 12500 REM *** Bedingter Sprung ***
[174] 12501 WORT=&H2158:GOSUB 11040
[084] 12502 WORT=&H75C0:GOSUB 11040
[F94] 12503 WORT=&HE903:GOSUB 11040
[FT3] 12504 WORT=ADI:GOTO 12000
[CK6] 12600 REM *** Variablen INPUT/READ lesen ***
[3A2] 12601 GOSUB 10070
[1K3] 12602 UP=&H24:GOSUB 11200
[ZD5] 12603 ON TYP GOTO 12604,12607,9980,12608
[9K3] 12604 UP=&H64:GOSUB 11200
[4K3] 12605 UP=&H51:GOSUB 11200
[V32] 12606 GOTO 12608
[FK3] 12607 UP=&H64:GOSUB 11200
[ZA2] 12608 GOSUB 13300
[376] 12609 IF EIN=44 THEN GOSUB 10030:GOTO 12601
```

```
[BW1] 12610 RETURN
[KR7] 12700 REM *** Adresse eines Strings auf Stapel **
*
[MA4] 12701 WORT=&HD88C:GOSUB 11040
[LF3] 12702 AUS=&H5:GOSUB 11000
[4W6] 12703 WORT=FNI(INT((VCI+15)/16)+ADR):GOSUB 11040
[6E3] 12704 AUS=&H50:GOTO 11000
[4A5] 12800 REM *** VARPTR auf Stapel ***
[BA2] 12801 GOSUB 10070
[0D4] 12802 IF TYP<>4 THEN RETURN
[YYC] 12803 IF INT((VCI+15)/16)+STRADR>4080 THEN PRINT
"String außer Reichweite";:GOTO 9990
[6M8] 12804 IF PASS=0 THEN STRPTR(PTRCNT)=FNI(VCI):VC!=
VCI+3
[4Y5] 12805 WORT=STRPTR(PTRCNT):GOSUB 11080
[GL3] 12806 UP=&H3C:GOSUB 11200
[1KG] 12807 PTRCNT=PTRCNT+1:IF PTRCNT>99 THEN PRINT "Zu
viele Stringparameter bei CALL/VARPTR";:GOTO 999
0
[2X1] 12808 RETURN
[BD3] 12900 REM *** AS lesen ***
[XU3] 12901 IF EIN<65 THEN 12904
[Y92] 12902 GOSUB 9995
[6L3] 12903 IF EIN=83 THEN 10030
[NT5] 12904 PRINT "AS erwartet";:GOTO 9990
[XM7] 13000 REM *** Zuweisung für LSET/RSET lesen ***
[A92] 13001 GOSUB 10030
[G92] 13002 GOSUB 10070
[PL3] 13003 IF TYP<>4 THEN 9980
[LU7] 13004 IF EIN<231 THEN PRINT "'=' erwartet";:GOTO
9990
[K92] 13005 GOSUB 10030
[V62] 13006 GOSUB 6000
[YL3] 13007 IF TYP<>4 THEN 9980
[5W1] 13008 RETURN
[0R6] 13100 REM *** Stringkonstante schreiben ***
[Y94] 13101 WORT=&HE80E:GOSUB 11040
[XY4] 13102 WORT=LEN(VAR$)+1:GOSUB 11040
[0H4] 13103 AUS=LEN(VAR$):GOSUB 11000
[F1A] 13104 FOR I=1 TO LEN(VAR$):AUS=ASC(MID$(VAR$,I,1)
):GOSUB 11000:NEXT
[5A3] 13105 UP=&H12:GOTO 11200
[T28] 13200 REM *** Zeilennummer für GOTO/GOSUB lesen *
**
[XF3] 13201 GOSUB 9995:X1=EIN
[864] 13202 GOSUB 9995:X1=X1+EIN*256
[8P5] 13203 IF PASS=0 THEN WORT=0:GOTO 10030
[W21] 13204 A=0
[EA2] 13205 B=ZEILEN-1
[F22] 13206 I=(A+B)\2
[ZL3] 13207 Z1=FNRI(ZEILNR(I))
[FW6] 13208 IF Z1=X1 THEN WORT=ZEILADR(I):GOTO 10030
[829] 13209 IF I=A THEN PRINT "Undefinierte Zeilennumme
r";:GOTO 9990
[EU4] 13210 IF Z1<X1 THEN A=I:GOTO 13206
[YP2] 13211 B=I:GOTO 13206
[P55] 13300 REM *** Variable zuweisen ***
[FC5] 13301 ON TYP GOTO 13302,13304,9980,13308
[B94] 13302 WORT=&H5B58:GOSUB 11040
[S23] 13303 WORT=&H789:GOTO 11040
[B94] 13304 WORT=&H5A58:GOSUB 11040
[A94] 13305 WORT=&H895B:GOSUB 11040
[H84] 13306 WORT=&H8907:GOSUB 11040
[323] 13307 WORT=&H257:GOTO 11040
[NA3] 13308 UP=&H13:GOTO 11200
[GS5] 13400 REM *** 2 x Single nach Integer ***
[RH3] 13401 UP=&H51:GOSUB 11200
[2K3] 13402 UP=&H65:GOSUB 11200
[3W2] 13403 TYPEN(TP-1)=1
[DW1] 13404 RETURN
[US6] 13500 REM *** Zeichen aus Bibliothek lesen ***
[F26] 13501 IF Z3>127 THEN GET #3,I3:I3=I3+1:Z3=0
[ZT1] 13502 Z3=Z3+1
[BB4] 13503 EIN=ASC(MID$(EIN3$,Z3,1))
[HW1] 13504 RETURN
```

Listing 1. Der Quelltext des SBasic-Compilers »sbasic.bas«


```
[B52] 1: CODE SEGMENT
[E22] 2: ASSUME CS:CODE
[900] 3:
[4R7] 4: ;Adressen der einzelnen Unterprogramme, die
se
[VX7] 5: ;werden je nach Bedarf zur EXE-Datei dazuge
bunden
[900] 6:
[308] 7: DW L00,L01,L02,L03,L04,L05,L06,L07,L
08,L09,L10,L11,L12,L13,L14,L15,L16,L17,L
18,L19,L20,L21,L22,L23,L24,L25,L26,L27,L
28,L29,L30,L31,L32,L33,L34,L35,L36,L37,L
38,L39,L40,L41,L42,L43,L44,L45,L46,L47,L
48,L49,L50,L51,L52,L53,L54,L55,L56,L57,L
58,L59,L60,L61,L62,L63,L64,L65,L66,L67,L
68,L69,L70,L71,L72,L73,L74,L75,L76,L77,L
78,L79,L80,L81,L82,L83,L84,L85,L86,L87,L
88,L89,L90,L91,L92,L93,L94,L95,L96,L97,L
98,L99
[WM0] 17: DW -1
[900] 18:
[2M6] 19: ;Das erste Modul bildet eine Ausnahme, es
enthält den Initialisierungscode und wird
[UF7] 20: ;immer mit eingebunden. Deshalb sind hier
[AT7] 21: ;einige wichtige Unterprogramme unterge-
[2G7] 22: ;bracht, die von anderen Modulen aufgerufen
[BR2] 23: ;werden können
[2H0] 24: ;
[2H0] 25: ;
[2H0] 26: ;PSP-Adresse sichern
[7V1] 27: MOV BP,DS
[7H3] 28: ;Datensegment sichern
[7V1] 29: MOV DX,CS
[7V1] 30: ADD AX,DX
[7V1] 31: MOV DS,AX
[3H4] 32: ;Variablen mit Vorzeichen
[1W1] 33: MOV ES,AX
[1W1] 34: AND BX,BX
[691] 35: JE X001
[9H2] 36: XOR AX,AX
[6U1] 37: XOR DI,DI
[3G1] 38: MOV CX,8
[602] 39: REP STOSW
[6V1] 40: MOV AX,ES
[451] 41: INC AX
[1W1] 42: MOV ES,AX
[341] 43: DEC BX
[KE1] 44: JNE X000
[7B3] 45: X001: MOV DS:[42],BP
[MB7] 46: ;Pseudo-Handle für "PRINT" und "INPUT"
[WN3] 47: MOV BYTE PTR DS:[0],-1
[E43] 48: ;Handle "LPRINT"
[BN3] 49: MOV BYTE PTR DS:[4],4
[FL2] 50: ;Aktuelle Farbe
[WN3] 51: MOV BYTE PTR DS:[9],7
[7S6] 52: ;Position des letzten Zeichens in Zeile
[LT1] 53: MOV AH,0FH
[681] 54: INT 10H
[F31] 55: DEC AH
[602] 56: MOV DS:[11],AH
[626] 57: ;Untere Grenze für "VIEW PRINT"
[2U3] 58: MOV BYTE PTR DS:[13],23
[L49] 59: ;Segment für "PEEK"/"POKE"/"CALL (Def Seg)"
[5H2] 60: MOV DS:[25],DS
[W33] 61: ;Cursorform holen
[7R1] 62: MOV AH,03H
[681] 63: INT 10H
[2H2] 64: MOV DS:[15],CX
[353] 65: ;Input-Flag setzen
[PW3] 66: MOV WORD PTR DS:[21],-1
[H8A] 67: ;Scrollen, wenn Cursor in Zeile 25 (Standard
d- Fenster ist 1 TO 24)
[3M1] 68: CMP DH,24
[091] 69: JB X002
[142] 70: MOV AX,0E0AH
[681] 71: INT 10H
[HN1] 72: MOV DH,23
[3R1] 73: MOV AH,02H
[681] 74: INT 10H
[KK3] 75: ;Cursor ausschalten
[WU2] 76: X002: MOV CX,1F0FH
[2P1] 77: MOV AH,01H
[681] 78: INT 10H
[L65] 79: ;Aktuelle Cursorform berechnen
[C25] 80: ;(Monochromadapter/Herules/CGA)
[D22] 81: ;und setzen
[122] 82: MOV DX,0707H
[LT1] 83: MOV AH,0FH
[681] 84: INT 10H
[UD1] 85: CMP AL,7
[621] 86: JNE U000
[522] 87: MOV DX,0B0CH
[D33] 88: X000: MOV DS:[36],DX
[C03] 89: ;Startwert für "RND"
[LD4] 90: MOV WORD PTR DS:[38],8E4AH
[X04] 91: MOV WORD PTR DS:[36],DX
[A33] 92: ;Daten für "PLAY"
[D84] 93: MOV WORD PTR DS:[44],1072
[H84] 94: MOV WORD PTR DS:[46],1912
[3P0] 95: RET
[900] 96:
[9K4] 97: ;Speicher für serielle Puffer
[GU2] 98: U001 DB 64 DUP (0)
[HU2] 99: U002 DB 64 DUP (0)
[FR1] 100: U003 DW 0,0
[GR1] 101: U004 DW 0,0
[7F1] 102: U005 DB 0
[8F1] 103: U006 DB 0
[9F1] 104: U007 DB 0
[AF1] 105: U008 DB 0
[900] 106:
[5T4] 107: ;Interruptroutine "COM1"
[X82] 108: U009: PUSH AX
[V91] 109: PUSH DX
[CR4] 110: ;Ist ein Zeichen vorhanden?
[132] 111: MOV DX,03FDH
[CM1] 112: IN AL,DX
[1E1] 113: SHR AL,1
[MF1] 114: JNC U00B
[M24] 115: ;Zeichen vom Port lesen
[3Y1] 116: MOV DL,0F8H
[CM1] 117: IN AL,DX
[L91] 118: PUSH BX
[7G3] 119: MOV BL,CS:[U005-L00]
[851] 120: INC BX
[2L1] 121: AND BX,63
[RC3] 122: ;Eingabepuffer voll?
[RE3] 123: CMP BL,CS:[U006-L00]
[AA1] 124: JE U00A
[Y36] 125: ;Zeichen in Eingabepuffer schreiben
[8T3] 126: MOV CS:[U001-L00+BX],AL
[H35] 127: ;Eingabepuffer aktualisieren
[YF3] 128: MOV CS:[U005-L00],BL
[L32] 129: U00A: POP BX
[1W4] 130: ;Bestätigung an Controller
[4N2] 131: U00B: MOV AL,64H
[9S1] 132: OUT DX,AL
[N71] 133: POP DX
[771] 134: POP AX
[CR0] 135: IRET
[900] 136:
[U73] 137: ;Interrupt "COM2"
[792] 138: U00C: PUSH AX
```

```
[V91] 139: PUSH DX
[X22] 140: MOV DX,02FDH
[CM1] 141: IN AL,DX
[1E1] 142: SHR AL,1
[XF1] 143: JNC U00E
[3Y1] 144: MOV DL,0F8H
[CM1] 145: IN AL,DX
[LA1] 146: PUSH BX
[8G3] 147: MOV BL,CS:[U007-L00]
[851] 148: INC BX
[1E1] 149: AND BX,63
[VE3] 150: CMP CS:[U008-L00]
[LA1] 151: JE U00D
[BT3] 152: MOV CS:[U002-L00+BX],AL
[4G3] 153: MOV CS:[U007-L00],BL
[P32] 154: U00D: POP BX
[4N2] 155: U00E: MOV AL,63H
[9S1] 156: OUT 0F8H,AL
[N71] 157: POP DX
[971] 158: POP AX
[GR0] 159: IRET
[900] 160:
[H17] 161: ;Zeichen in eine Diskettendatei schreiben
[E22] 162: X003: CMP AL,13
[P91] 163: JE X004
[8L1] 164: CMP CL,10
[K91] 165: JE X005
[S85] 166: ;Positionseiger weiterzählen
[E33] 167: INC BYTE PTR DS:[4+BX]
[8G2] 168: JNE SHORT X005
[AP7] 169: ;Bei "Carriage Returns" oder "Line Feeds"
[KV3] 170: ;Positionseiger auf 0
[4Y4] 171: X004: MOV BYTE PTR DS:[4+BX],0
[5V3] 172: ;Datei-Handle holen
[WH3] 173: X005: MOV BL,DS:[BX]
[EF1] 174: MOV CX,1
[H43] 175: ;Zeichen in Puffer
[DC2] 176: MOV DS:[14],AL
[171] 177: MOV DX,14
[NW5] 178: ;Puffer (1 Byte) in Datei schreiben
[7R1] 179: MOV AH,04H
[NB1] 180: INT 21H
[D71] 181: POP BX
[3P0] 182: RET
[896] 183: ;Ein Zeichen in die Datei BX schreiben
[392] 184: X006: PUSH BX
[E62] 185: ;Handle holen
[UP2] 186: MOV CL,DS:[BX]
[BD4] 187: ;Ist es ein Pseudo-Handle?
[GL1] 188: CMP CL,-3
[891] 189: JB X007
[P91] 190: PUSH AX
[909] 191: ;Pseudo-Handle -2 = "COM1"
[CL1] 192: CMP CL,-2
[8G1] 193: JNE SHORT X007
[543] 194: ;Ausgabepuffer leer?
[132] 195: MOV DX,03FDH
[9B2] 196: U007: IN AL,DX
[KR1] 197: TEST AL,2FH
[WAL] 198: JE X00C
[RG4] 199: ;Zeichen in Port schreiben
[3Y1] 200: MOV DL,0F8H
[771] 201: POP AX
[0V1] 202: MOV DX,AL
[D71] 203: POP BX
[3P0] 204: RET
[D05] 205: ;Pseudo-Handle -3 = "COM2"
[XA2] 206: U008: CMP CL,-3
[EG1] 207: JNE U008
[X22] 208: MOV DX,02FDH
[B92] 209: U009: IN AL,DX
[KR1] 210: TEST AL,2FH
[2B1] 211: JE X007
[3Y1] 212: MOV DL,0F8H
[771] 213: POP AX
[0V1] 214: MOV DX,AL
[D71] 215: POP BX
[3P0] 216: RET
[G25] 217: ;Pseudo-Handle -1 = "SCRN"
[472] 218: U009: MOV AH,0FH
[WAL] 219: JE X007
[4W3] 220: ;Cursorposition holen
[7R1] 221: MOV AH,03H
[681] 222: INT 10H
[7H1] 223: POP CX
[XW1] 224: ;Farbe holen
[KL2] 225: MOV BL,DS:[9]
[GR2] 226: ;Beep ((Ctrl-G))
[2E1] 227: CMP CL,7
[7F1] 228: JNE X007
[HV4] 229: ;Beep über das BIOS ausgeben
[D32] 230: MOV AX,0E07H
[681] 231: INT 10H
[D71] 232: POP BX
[3P0] 233: RET
[F62] 234: X007: CMP CL,9
[DF1] 235: JNE X009
[AN3] 236: ;Tabulator ((Ctrl-I))
[2K1] 237: CALL X00A
[4W3] 238: ;Cursorposition holen
[7R1] 239: MOV AH,03H
[681] 240: INT 10H
[7K6] 241: ;Anzahl der Leerzeichen zum nächsten
[7W3] 242: ;Tabulator nach CX
[3U1] 243: MOV AL,DL
[9C1] 244: AND AL,7
[3G1] 245: MOV CX,8
[GS1] 246: SUB CL,AL
[L74] 247: ;CX Leerzeichen ausgeben
[H12] 248: MOV AX,0920H
[681] 249: INT 10H
[LP4] 250: ;Return-Flag zurücksetzen
[HR3] 251: MOV BYTE PTR DS:[10],0
[F85] 252: ;Neue Cursorposition berechnen
[BP1] 253: ADD DL,CL
[ME6] 254: ;Cursor am rechten Rand, dann neue Zeile
[XE2] 255: CMP DL,DS:[11]
[6E1] 256: JBE X008
[31] 257: DEC DL
[9P3] 258: ;Return-Flag setzen
[N3] 259: DEC BYTE PTR DS:[10]
[LL2] 260: ;Cursor setzen
[YL2] 261: X008: MOV AH,02H
[681] 262: INT 10H
[D71] 263: POP BX
[3P0] 264: RET
[YA8] 265: ;"Line Feed" und "Carriage Returns" ((Ctrl-J
))
[MS1] 266: ;[Enter]]
[HA2] 267: X009: CMP CL,10
[PA1] 268: JE X00A
[WL1] 269: CMP CL,13
[8G1] 270: JNE X009
[AA3] 271: ;In neue Zeile gehen
[6B2] 272: X00A: CALL X00P
[D71] 273: POP BX
[3P0] 274: RET
[4Y4] 275: ;"Cursor Home" ((Ctrl-K))
[XA2] 276: X00B: CMP CL,11
[EG1] 277: JNE X00D
[S73] 278: ;An den linken Rand
[C72] 279: X00C: MOV DL,0
[SAS] 280: ;und an die oberste Zeile gehen
[P22] 281: MOV DH,DS:[12]
[LL2] 282: ;Cursor setzen
[3R1] 283: MOV AH,02H
[681] 284: INT 10H
[8R4] 285: ;Returnflag zurücksetzen
[HR3] 286: MOV BYTE PTR DS:[10],0
[D71] 287: POP BX
```

```
[3P0] 288: RET
[KAS] 289: ;"Clear Screen" ((Ctrl-L))
[2B2] 290: X00D: CMP CL,12
[HG1] 291: JNE X00E
[SP5] 292: ;Bildschirm über das BIOS löschen
[312] 293: MOV 0,0000H
[ER1] 294: CALL X005
[423] 295: ;"Cursor Home"
[WH1] 296: JNE X00C
[4U2] 297: ;Cursor rechts?
[R82] 298: X00E: CMP CL,28
[RG1] 299: JNE X00C
[9S2] 300: ;Position holen
[7R1] 301: MOV AH,03H
[8L1] 302: INT 10H
[4W2] 303: ;Am rechten Rand?
[XE2] 304: CMP DL,DS:[11]
[5B1] 305: JNE X00F
[451] 306: INC DL
[WK2] 307: JNE SHORT X00I
[AA3] 308: ;In neue Zeile gehen
[FT2] 309: X00F: MOV DL,0
[VK2] 310: JNE SHORT X00H
[2L2] 311: ;Cursor unten?
[6B2] 312: X00G: CMP CL,31
[3H1] 313: JNE X00K
[9S2] 314: ;Position holen
[7R1] 315: MOV AH,03H
[8L1] 316: INT 10H
[222] 317: ;Am unteren Rand?
[MC3] 318: X00H: CMP DH,DS:[13]
[HB1] 319: JNE X00J
[951] 320: INC DH
[LL2] 321: ;Cursor setzen
[FM2] 322: X00I: MOV AH,02H
[7R1] 323: INT 10H
[HR3] 324: MOV BYTE PTR DS:[10],0
[242] 325: X00J: POP BX
[3P0] 326: RET
[CH2] 327: ;Cursor links?
[QC2] 328: X00K: CMP CL,29
[6H1] 329: JNE X00L
[9S2] 330: ;Position holen
[7R1] 331: MOV AH,03H
[8L1] 332: INT 10H
[4M2] 333: ;Am linken Rand?
[HD1] 334: CMP DL,0
[XB1] 335: JE X00N
[231] 336: DEC DL
[6K1] 337: JNE X00I
[7G4] 338: ;Eine Zeile nach oben gehen
[FG2] 339: MOV DL,DS:[11]
[6L2] 340: JNE SHORT X00M
[492] 341: ;Cursor oben?
[8B2] 342: X00L: CMP CL,30
[CH1] 343: JNE X00N
[9S2] 344: ;Position holen
[7R1] 345: MOV AH,03H
[8L1] 346: INT 10H
[SK2] 347: ;Am oberen Rand?
[PC3] 348: X00M: CMP DH,DS:[12]
[HB1] 349: JNE X00J
[OUT] 350: DEC DH
[V31] 351: JNE X00I
[6K1] 352: JNE X00I
[777] 353: ;Druckbare Zeichen über das BIOS ausgeben
[HB2] 354: X00N: CALL X000
[BU1] 355: MOV AL,CL
[2R1] 356: MOV AH,09H
[EF1] 357: MOV CX,1
[681] 358: INT 10H
[EP4] 359: ;Ein Zeichen nach rechts gehen
[MK1] 360: MOV DX,0
[442] 361: X00O: POP BX
[3P0] 362: RET
[344] 363: ;Eventuell in neue Zeile gehen
[SD3] 364: X00P: CMP DL,DS:[11]
[YH1] 365: JNE X00T
[MF3] 366: ;Returnflag abfragen
[EN3] 367: CMP BYTE PTR DS:[10],0
[FC1] 368: JE X00T
[AA3] 369: ;In neue Zeile gehen
[AA2] 370: X00P: PUSH CX
[553] 371: ;Cursor nach links
[3P1] 372: MOV DL,0
[X04] 373: ;Cursor am unteren Rand?
[LF2] 374: CMP DH,DS:[13]
[NH1] 375: JNE X00Q
[WT5] 376: ;Ausrechnen, wie viele Zeichen nach scrollen
[512] 377: MOV AX,0F01H
[EK1] 378: CALL X00S
[5L2] 379: JNE SHORT X00R
[G95] 380: ;Cursor ein Zeichen nach unten
[X02] 381: X00Q: INC DH
[VW2] 382: ;Position setzen
[SM2] 383: X00R: MOV AH,02H
[681] 384: INT 10H
[KS3] 385: ;Returnflag löschen
[HR3] 386: MOV BYTE PTR DS:[10],0
[3P0] 387: POP CX
[9A8] 388: ;Ausschnitt scrollen/löschen, je nach Wert
in AX
[9A2] 389: X00S: PUSH BX
[891] 390: PUSH CX
[V91] 391: PUSH DX
[LL1] 392: ;Attribut
[XT1] 393: MOV BH,BL
[V21] 394: ;Links
[2E1] 395: MOV CL,0
[VCI] 396: ;Rechts
[8G2] 397: MOV CH,DS:[12]
[C09] 398: ;Oben
[FG2] 399: MOV DL,DS:[11]
[N31] 400: ;Unten
[L02] 401: MOV DH,DS:[13]
[681] 402: INT 10H
[N71] 403: POP DX
[771] 404: POP CX
[D71] 405: POP BX
[9C1] 406: X00T: RET
[499] 407: ;Cursor ein Zeichen nach rechts nach Zeiche
nausgabe
[YM2] 408: X00U: MOV AH,03H
[681] 409: INT 10H
[4W2] 410: ;Am rechten Rand?
[XE2] 411: CMP DL,DS:[11]
[NCI] 412: JNE X00V
[451] 413: INC DL
[VW2] 414: ;Position setzen
[3R1] 415: MOV AH,02H
[681] 416: INT 10H
[KS3] 417: ;Returnflag löschen
[HR3] 418: MOV BYTE PTR DS:[10],0
[3P0] 419: RET
[XV6] 420: ;Cursor nicht bewegen, Returnflag setzen
[PL4] 421: X00V: MOV BYTE PTR DS:[10],-1
[3P0] 422: RET
[900] 423:
[4H5] 424: ;Ein Zeichen aus DATA-Zeile lesen
[UC2] 425: X00W: MOV AL,26
[AN3] 426: ;Positionseiger = 0?
[RG2] 427: MOV DI,DS:[23]
[XP1] 428: AND DI,DI
[9W8] 429: ;Ja, dann Leerstring lesen, entspricht "Out
of Data"
[9C1] 430: JE U00V
[LGS] 431: ;Zeichen aus Programmtext lesen
[TM2] 432: MOV AL,CS:[DI]
[S25] 433: ;Positionseiger weiterzählen
[151] 434: INC DI
```



```
[D29] 435: ;Bei Ende Positionszeiger auf nächste DATA-
[3M1] 436: CMP AL,26
[WH1] 437: JNE Y00V
[2N2] 438: MOV DI,CS:[DI]
[2C3] 439: Y00V: MOV DS:[23],DI
[WB1] 440: Y00V: RET
[945] 441: ;Ein Zeichen aus Datei BX lesen
[KG2] 442: AND BX,BX
[OD5] 443: ;BH = -1, dann aus DATA-Zeile lesen
[VD1] 444: Y00V: JNE Y00P
[EE2] 445: ;Handle holen
[KP2] 446: MOV AL,DS:[BX]
[W34] 447: ;Bildschirm (<SCRN:*)?
[OL1] 448: CMP AL,-1
[DL1] 449: JE Y00Z
[L91] 450: PUSH BX
[US6] 451: ;Serielle Schnittstelle <COM1:~?
[AL1] 452: CMP AL,-2
[7S1] 453: JNE X00W0
[946] 454: ;Leseoperation im Eingabepuffer holen
[G54] 455: X00W1: MOV BL,CS:[Y006-L00]
[INS] 456: ;Mit Schreibposition vergleichen
[NE3] 457: CMP BL,CS:[Y005-L00]
[226] 458: ;Kein Zeichen vorhanden, dann warten
[WK1] 459: JE X00W1
[W41] 460: INC BL
[PK1] 461: AND BL,63
[OD2] 462: ;Zeichen lesen
[E3V] 463: MOV AL,CS:[Y001-L00+BX]
[UW4] 464: ;Leseoperation weiterzählen
[IG3] 465: MOV CS:[Y006-L00],BL
[DL1] 466: POP BX
[S43] 467: MOV DS:[16+BX],BH
[3P0] 468: RET
[W56] 469: ;Serielle Schnittstelle <COM2:~?
[4D2] 470: X00W2: CMP AL,-3
[7S1] 471: JNE X00W2
[PS4] 472: X00W3: MOV BL,CS:[Y008-L00]
[TE3] 473: CMP BL,CS:[Y007-L00]
[4L1] 474: JE X00W3
[W41] 475: INC BL
[PK1] 476: AND BL,63
[WK1] 477: MOV AL,CS:[Y002-L00+BX]
[7G3] 478: MOV CS:[Y008-L00],BL
[DL1] 479: POP BX
[S43] 480: MOV DS:[16+BX],BH
[3P0] 481: RET
[C54] 482: ;Normale Diskettendatei
[D62] 483: X00W2: POP BX
[W56] 484: ;Letzte gelesene Zeichen holen
[D73] 485: MOV CL,DS:[16+BX]
[R91] 486: PUSH CX
[7N5] 487: ;Mit Dateiendezeichen vergleichen
[EM1] 488: CMP CL,26
[VC1] 489: JE X00Z
[L91] 490: PUSH BX
[TK2] 491: ;Handle nach BX
[VT1] 492: MOV BL,AL
[EF1] 493: MOV CX,1
[2P1] 494: MOV DX,14
[R56] 495: ;Ein Zeichen aus Datei in Puffer lesen
[WT1] 496: MOV AH,3FH
[N81] 497: INT 21H
[DL1] 498: POP BX
[WC3] 499: ;Fehler, dann Dateiende
[MC1] 500: JC X00Z
[L51] 501: AND AX,AX
[FG2] 502: MOV AL,DS:[14]
[DK1] 503: JNE X00Y
[293] 504: ;Dateiende erreicht
[VC2] 505: X00X: MOV AL,26
[ML3] 506: ;Neues Zeichen sichern
[AX3] 507: X00Y: MOV DS:[16+BX],AL
[971] 508: X00X: RET
[3P0] 509: RET
[266] 510: ;Von Bildschirm oder Tastatur lesen
[GE3] 511: X00Z: MOV AL,DS:[14]
[OL1] 512: CMP AL,-1
[PE1] 513: JNE Y000
[XK2] 514: ;Tastatur (INPUS)
[VR1] 515: MOV AH,08H
[N81] 516: INT 21H
[3P0] 517: RET
[N83] 518: ;Eingabeflag prüfen
[AK4] 519: Y000: MOV WORD PTR DS:[21],-1
[G91] 520: JE Y003
[BV8] 521: ;Ende der Eingabezeile, dann Endezeichen zu-
rückgeben
[5D1] 522: CMP AL,0
[TE1] 523: JNE Y001
[NN1] 524: MOV AL,26
[3P0] 525: RET
[S46] 526: ;Anzahl der Zeichen um eins verringern
[S44] 527: Y001: DEC BYTE PTR DS:[14]
[L91] 528: PUSH BX
[YM3] 529: ;Bildschirmseite holen
[L71] 530: MOV AH,0FH
[NG1] 531: INT 10H
[OD2] 532: ;Zeichen lesen
[VR1] 533: MOV AH,08H
[G81] 534: INT 10H
[F91] 535: PUSH AX
[4W3] 536: ;Cursorposition holen
[7R1] 537: MOV AH,03H
[G81] 538: INT 10H
[U55] 539: ;Cursor um ein Zeichen nach rechts
[XE2] 540: CMP DL,DS:[11]
[ND1] 541: JAE Y002
[451] 542: INC DL
[VW2] 543: ;Position setzen
[3R1] 544: MOV AH,02H
[G81] 545: INT 10H
[932] 546: Y002: POP AX
[D71] 547: POP BX
[3P0] 548: RET
[353] 549: ;Zeile eingeben lassen
[292] 550: Y003: PUSH BX
[F21] 551: ;Seite holen
[L91] 552: MOV AH,0FH
[G81] 553: INT 10H
[4W3] 554: ;Cursorposition holen
[7R1] 555: MOV AH,03H
[G81] 556: INT 10H
[V13] 557: ;Cursorform sichern
[2H2] 558: MOV DS:[21],CX
[V91] 559: PUSH DX
[AZ2] 560: ;Insert-Modus aus
[G01] 561: XOR DI,DI
[V54] 562: ;Zeilenende überschritten?
[CC3] 563: Y004: CMP DL,DS:[11]
[L81] 564: JBE Y005
[FG2] 565: MOV DL,DS:[11]
[S44] 566: ;Cursorposition setzen
[X12] 567: Y005: MOV AH,02H
[G81] 568: INT 10H
[DH3] 569: ;Cursor einschalten
[HH2] 570: MOV CX,DS:[36]
[TH3] 571: ;Insert-Flag prüfen
[XP1] 572: AND DI,DI
[T91] 573: JE Y006
[TR3] 574: ;Dicker Insert-Cursor
[L81] 575: MOV AH,04H
[IC3] 576: ;Cursorform setzen
[V12] 577: Y006: MOV AH,01H
[G81] 578: INT 10H
[C11] 579: ;Taste holen
[VR1] 580: MOV AH,08H
[N81] 581: INT 21H
[F91] 582: PUSH AX
```

```
[XK3] 583: ;Cursor ausschalten
[E32] 584: MOV CX,Y00H
[ZP1] 585: MOV AH,01H
[G81] 586: INT 10H
[971] 587: POP AX
[UL4] 588: ;Ist es ein Steuerzeichen?
[XG1] 589: CMP AL," "
[MA1] 590: JB Y00B
[W55] 591: ;Nein, dann aktuelle Farbe lesen
[V63] 592: Y007: MOV BL,DS:[9]
[VW2] 593: ;Insert-Modus an?
[XP1] 594: AND DI,DI
[UAI] 595: JE Y00A
[AAR] 596: ;Rest der Zeile um ein Zeichen nach rechts
schieben
[F91] 597: PUSH AX
[BUI] 598: MOV CL,DL
[FG2] 599: MOV DL,DS:[11]
[LL2] 600: ;Ende erreicht?
[L92] 601: Y008: DEC DI
[C81] 602: JS Y009
[US1] 603: CMP DL,CL
[791] 604: JB Y009
[S44] 605: ;Cursorposition setzen
[3R1] 606: MOV AH,02H
[G81] 607: INT 10H
[OD2] 608: ;Zeichen lesen
[VR1] 609: MOV AH,08H
[G81] 610: INT 10H
[PK5] 611: ;Cursor ein Zeichen nach rechts
[451] 612: INC DL
[3R1] 613: MOV AH,02H
[G81] 614: INT 10H
[R91] 615: PUSH CX
[HW2] 616: ;Zeichen ausgeben
[EF1] 617: MOV CX,1
[ER1] 618: MOV AH,09H
[G81] 619: INT 10H
[W17] 620: POP CX
[S55] 621: ;Cursor um zwei Zeichen nach links
[Z31] 622: DEC DL
[RG1] 623: JNE Y008
[XK2] 624: ;Cursor wieder auf Position setzen
[EF2] 625: Y009: MOV DL,CL
[3R1] 626: MOV AH,02H
[G81] 627: INT 10H
[971] 628: POP AX
[HW2] 629: ;Zeichen ausgeben
[V82] 630: Y00A: MOV CX,1
[ER1] 631: MOV AH,09H
[G81] 632: INT 10H
[FR3] 633: ;Cursor nach rechts
[451] 634: INC DL
[RG1] 635: JMP Y004
[XK2] 636: ;Steuerzeichen
[ML2] 637: Y00B: CMP AL,00H
[4H1] 638: JNE Y00J
[3U4] 639: ;Erweiterten Scancode holen
[VR1] 640: MOV AH,08H
[N81] 641: INT 21H
[TH2] 642: ;Cursor links?
[LS1] 643: CMP AL,4BH
[RG1] 644: JNE Y00D
[4M2] 645: ;Am linken Rand?
[971] 646: POP AX
[F91] 647: PUSH AX
[MS1] 648: CMP DL,AL
[BF1] 649: JBE Y00C
[Z31] 650: DEC DL
[G52] 651: Y00C: JMP Y004
[4U2] 652: ;Cursor rechts?
[UN2] 653: Y00D: CMP AL,4DH
[SG1] 654: JNE Y00F
[S12] 655: Y00E: INC DL
[SH1] 656: JMP Y00C
[F93] 657: ;Cursor <home>?
[NM2] 658: Y00F: CMP DL,47H
[VGI] 659: JNE Y00G
[183] 660: ;An den Anfang gehen
[WT1] 661: POP DX
[V91] 662: PUSH CX
[SH1] 663: JMP Y00C
[2X2] 664: ;Cursor <end>?
[3P2] 665: Y00G: CMP AL,4FH
[1H1] 666: JNE Y00I
[HM3] 667: ;Ans Eingabeende gehen
[FG2] 668: MOV DL,DS:[11]
[WT1] 669: POP CX
[R91] 670: PUSH CX
[LL2] 671: ;Cursor setzen
[GM2] 672: Y00H: MOV AH,02H
[G81] 673: INT 10H
[OD2] 674: ;Zeichen lesen
[VR1] 675: MOV AH,08H
[G81] 676: INT 10H
[R36] 677: ;Kein Leerzeichen, dann Ende gefunden
[XG1] 678: CMP AL," "
[NG1] 679: JNE Y00E
[3U3] 680: ;Linker Rand erreicht?
[US1] 681: CMP DL,CL
[G81] 682: JE Y00C
[ZL4] 683: ;Nächstes Zeichen prüfen
[Z31] 684: DEC DL
[7K1] 685: JMP Y00H
[D72] 686: ;Delete?
[EM2] 687: Y00I: CMP AL,53H
[SB1] 688: JE Y00K
[E82] 689: ;Insert?
[4P1] 690: CMP AL,52H
[FG1] 691: JNE Y00C
[R54] 692: ;Insert-Flag ausschalten
[R71] 693: NOT DI
[SH1] 694: JMP Y00C
[UK2] 695: ;Backspace?
[NM2] 696: Y00J: CMP AL,08H
[LH1] 697: JNE Y000
[4M2] 698: ;Am linken Rand?
[WT1] 699: POP CX
[R91] 700: PUSH CX
[US1] 701: CMP DL,CL
[CO1] 702: JBE Y00N
[P85] 703: ;Cursor ein Zeichen nach links
[Z31] 704: DEC DL
[V14] 705: ;Rest der Zeile nachrücken
[9A2] 706: Y00K: PUSH DX
[KL2] 707: MOV BL,DS:[9]
[4W2] 708: ;Am rechten Rand?
[4D3] 709: Y00L: CMP DL,DS:[11]
[G61] 710: JMP Y00M
[EP3] 711: ;Cursor nach rechts
[451] 712: INC DL
[3R1] 713: MOV AH,02H
[G81] 714: INT 10H
[OD2] 715: ;Zeichen lesen
[VR1] 716: MOV AH,08H
[G81] 717: INT 10H
[S53] 718: ;Cursor nach links
[Z31] 719: DEC DL
[3R1] 720: MOV AH,02H
[G81] 721: INT 10H
[V33] 722: ;Zeichen schreiben
[EF1] 723: MOV CX,1
[Z11] 724: MOV AH,09H
[G81] 725: INT 10H
[0A5] 726: ;Nächstes Zeichen verschieben
[451] 727: INC DL
[LK1] 728: JMP Y00L
[4M2] 729: ;Cursor am Ende
[NM2] 730: Y00M: MOV AH,02H
[G81] 731: INT 10H
```

```
[LE4] 732: ;Leerzeichen nachrücken
[H12] 733: MOV AX,0920H
[EF1] 734: MOV CX,1
[G81] 735: INT 10H
[N71] 736: POP DX
[V52] 737: Y00N: JMP Y004
[CY3] 738: ;Carriage Return?
[CB2] 739: Y000: CMP AL,13
[TC1] 740: JE Y00P
[NH8] 741: ;Line Feed überlesen (für Input-Redirecti-
on)
[8L1] 742: CMP AL,10
[IC1] 743: JE Y00N
[MGI] 744: JMP Y007
[HPS] 745: ;Eingabe abschließen, Ende suchen
[D42] 746: Y00P: POP CX
[FG2] 747: MOV DL,DS:[11]
[LL2] 748: ;Cursor setzen
[TM2] 749: Y00Q: MOV AH,02H
[G81] 750: INT 10H
[OD2] 751: ;Zeichen lesen
[VR1] 752: MOV AH,08H
[G81] 753: INT 10H
[S95] 754: ;Kein Leerzeichen, dann fertig
[XG1] 755: CMP AL," "
[MH1] 756: JNE Y00R
[TY1] 757: ;Nach links
[Z31] 758: DEC DL
[3U3] 759: ;Linker Rand erreicht?
[RD1] 760: CMP DL,CL
[US1] 761: CMP DL,CL
[XG1] 762: JAE Y00Q
[Z14] 763: ;Zeilenlänge speichern
[K12] 764: Y00R: INC DL
[US1] 765: SUB DL,1
[TO2] 766: MOV DS:[14],DL
[Z16] 767: ;Cursor auf Anfang der Zeile setzen
[C16] 768: MOV DL,CL
[3R1] 769: MOV AH,02H
[G81] 770: INT 10H
[D71] 771: POP BX
[TL1] 772: JMP Y00Z
[Y000] 773: ;Bei Bildschirmangaben nach Lesen aller
Zeilen
[Z74] 775: ;Variablen in neue Zeile gehen
[1P4] 776: Y00S: CMP BYTE PTR DS:[BX],-1
[2K1] 777: JNE Y00T
[G95] 778: ;Carriage Return: ausgeben
[G81] 779: MOV AL,13
[RF1] 780: CALL X006
[455] 781: ;Cursorform wiederherstellen
[UC2] 782: MOV CX,DS:[21]
[ZP1] 783: MOV AH,01H
[G81] 784: INT 10H
[FX4] 785: ;Eingabeflag zurücksetzen
[86] 786: MOV WORD PTR DS:[21],-1
[S81] 787: Y00T: RET 2
[Y000] 788: RET
[ZD4] 789: ;Zwei Strings vergleichen
[8K2] 790: Y00U: MOV SI,SP
[AB1] 791: ADD BP,4
[KR4] 792: ;Längen der Strings holen
[D73] 793: MOV CL,SS:[BP+256]
[3R2] 794: MOV CH,SS:[BP]
[GW3] 795: ;Vergleichszeiger auf 0
[GUL] 796: MOV DI,DI
[U66] 797: ;Ende der Strings erreicht, dann gleich
[RF2] 798: Y00V: JCXZ Y00X
[Z73] 799: ;Länge 1 < Länge 2?
[DO1] 800: CMP SI,DI
[WCI] 801: JE Y00W
[S73] 802: ;Länge 2 < Länge 1?
[9D1] 803: CMP CH,0
[ZD1] 804: JE Y00Y
[151] 805: INC DI
[V31] 806: DEC CL
[R31] 807: DEC CH
[7B4] 808: ;Zwei Zeichen vergleichen
[AW3] 809: MOV AL,SS:[BP+256+DI]
[K43] 810: CMP AL,SS:[BP+DI]
[VR4] 811: ;Gleich, dann weiter prüfen
[TC1] 812: JE Y00V
[PC1] 813: JA Y00Y
[ED1] 814: ;Kleiner
[X72] 815: Y00W: MOV AL,2
[3P0] 816: RET
[V91] 817: ;Gleich
[W72] 818: Y00X: MOV AL,1
[3P0] 819: RET
[S51] 820: ;Größer
[382] 821: Y00Y: MOV AL,4
[3P0] 822: RET
[OD2] 823: RET
[BA7] 824: ;Zwei Singles (einfache Realzahlen) addiere
[UN2] 825: ADDS: MOV BP,SP
[9K5] 826: ;Zahl 1 nach SI:DX, Exponent in BL
[3P1] 827: MOV DL,0
[S13] 828: MOV DH,SS:[BP+6]
[R33] 829: MOV SI,SS:[BP+7]
[U13] 830: MOV BL,SS:[BP+9]
[EG5] 831: ;Zahl 2 nach DI:AX, Exponent in BH
[RE1] 832: MOV AL,0
[413] 833: MOV AH,SS:[BP+2]
[L13] 834: MOV DI,SS:[BP+3]
[E13] 835: MOV BH,SS:[BP+5]
[M95] 836: ;Zahl 1 < Zahl 2, dann vertauschen
[8T1] 837: CMP SI,BH
[LDI] 838: JAE Z000
[V31] 839: XCHG DX,AX
[XU1] 840: XCHG SI,DI
[KS1] 841: XCHG BL,BH
[SH3] 842: ;Zahl 2 > 0, dann fertig
[Z52] 843: Z000: CMP BH,0
[RA1] 844: JE Z007
[XU4] 845: ;Exponentendifferenz nach CX
[SU1] 846: MOV CL,BL
[3S1] 847: SUB CL,BH
[CG5] 848: ;Größer 24, dann nicht addieren
[3M1] 849: CMP CL,24
[M91] 850: JA Z007
[V21] 851: MOV CH,0
[P84] 852: ;Vorzzeichen vergleichen
[8B1] 853: PUSH SI
[ZM1] 854: XOR SI,DI
[XZ0] 855: PUSHF
[F33] 856: MOV SI,SS:[BP+2]
[HG5] 857: ;Mantisse in Integerform bringen
[RW1] 858: OR SI,0000H
[NUL] 859: OR DI,0000H
[GP3] 860: ;Exponent 2 angleichen
[5K1] 861: JCXZ Z002
[W62] 862: Z001: SHR DI,1
[R61] 863: RCR AX,1
[NH1] 864: LOOP Z001
[K44] 865: ;Vorzzeichen verschieden?
[OM1] 866: Z002: POPF
[TA1] 867: JS Z003
[E83] 868: ;Mantissen addieren
[LS1] 869: ADD DX,AX
[R81] 870: ADC SI,DI
[PP4] 871: ;Überlauf berücksichtigen
[4F1] 872: JNC Y005
[NF1] 873: RCR SI,1
[8E1] 874: RCR DX,1
```

Listing 2. Der Quelltext der SBasic-Laufzeitbibliothek »sbasic.asm«


```
[W41] 875: INC BL
[DK2] 876: JMP SHORT Z006
[S44] 877: ;Mantissen subtrahieren
[GG2] 878: SUB DX,AX
[HS1] 879: SBB SI,DI
[244] 880: ;Unterlauf abtesten?
[FA1] 881: JC Z009
[W62] 882: ;Ergebnis = 0?
[6P1] 883: JNE Z004
[W81] 884: AND AX,DX
[6P1] 885: JNE Z004
[CK2] 886: JMP SHORT Z005
[OC6] 887: ;Zweierkomplement der Mantisse bilden
[UT2] 888: Z006: NOT DX
[HB1] 889: ADD DX,1
[E91] 890: NOT SI
[RC1] 891: ADC SI,0
[866] 892: ;Vorzeichen des Ergebnisses umdrehen
[L74] 893: XOR BYTE PTR SS:[BP-1],128
[974] 894: ;Ergebnis linksbündig machen
[Z22] 895: AND SI,SI
[7B1] 896: JS Z006
[L81] 897: RCL SI,1
[XE1] 898: RCL SI,1
[R31] 899: DEC BL
[6P1] 900: JNE Z004
[A91] 901: ;Ergebnis ist 0, wenn ein Faktor 0 ist
[3B2] 902: Z005: XOR SI,SI
[HW1] 903: XOR DX,DX
[V21] 904: MOV BL,0
[YG4] 905: ;Vorzeichenflag vergessen
[971] 906: POP AX

[EX2] 907: JMP SHORT Z007
[886] 908: ;Vorzeichen in das Ergebnis schreiben
[F32] 909: POP AX
[F21] 910: OR AX,FFFFH
[CW1] 911: AX
[7B3] 912: ;Ergebnis auf Stack
[9P3] 913: MOV SS:[BP+6],DH
[R13] 914: MOV SS:[BP+7],SI
[413] 915: MOV SS:[BP+9],BL
[L21] 916: RET 4
[000] 917:

[F15] 918: ;Zwei Singles multiplizieren
[S22] 919: MULS: MOV BP,SP
[6Y5] 920: ;Ergebnis ist 0, wenn ein Faktor 0 ist
[KY3] 921: CMP BYTE PTR SS:[BP+9],0
[5A1] 922: JE GR00
[2Y3] 923: CMP BYTE PTR SS:[BP+5],0
[5A1] 924: JE GR00
[PC6] 925: ;Niederwertige Bytes multiplizieren
[C13] 926: MOV BL,SS:[BP+2]
[G13] 927: MOV BH,SS:[BP+6]
[X71] 928: MOV AL,BH
[T61] 929: MUL BH
[LW1] 930: MOV DI,AX
[G06] 931: ;Höherwertige Worte multiplizieren
[F33] 932: MOV SI,SS:[BP+7]
[U13] 933: MOV BP,SS:[BP+7]
[KM6] 934: ;Vorzeichen des Ergebnisses bestimmen
[GV1] 935: MOV AX,SI
[ZU1] 936: MOV AX,BP
[F21] 937: OR AX,FFFFH
[F91] 938: PUSH AX
[S85] 939: ;Mantissen in Integerformat bringen
[HW1] 940: OR SI,8000H
[MU1] 941: OR BP,8000H
[GV1] 942: MOV AX,SI
[L71] 943: MUL BP
[005] 944: ;Multiplikation Wort 1 x Byte 2
[VY1] 945: MOV AL,DX
[TW1] 946: XCHG SI,DX
[RE1] 947: MOV AL,0
[971] 948: MOV AH,BH
[HT1] 949: MUL DX
[Z53] 950: ;Zum Ergebnis addieren
[KR1] 951: ADD DI,AX
[Z51] 952: ADC CX,DX
[RC1] 953: ADC SI,0
[Z75] 954: ;Multiplikation Wort 2 x Byte 1
[RE1] 955: MOV AL,0
[TT1] 956: MOV AH,BL
[L71] 957: MUL BP
[Z53] 958: ;Zum Ergebnis addieren
[KR1] 959: ADD DI,AX
[Z51] 960: ADC CX,DX
[RC1] 961: ADC SI,0
[971] 962: POP AX
[HW1] 963: MOV BP,SP
[W13] 964: ;Exponenten holen
[K13] 965: MOV BL,SS:[BP+5]
[P13] 966: MOV BH,SS:[BP+9]
[3A5] 967: ;Exponenten ins Integerformat
[N12] 968: BX,8000H
[RH3] 969: ;Exponenten addieren
[GN1] 970: ADD BL,BH
[VH1] 971: JNO Z00C
[MA4] 972: ;Überlauf, Ergebnis ist 0
[NR2] 973: GR00: XOR AX,AX
[X13] 974: MOV SS:[BP+6],AX
[Z23] 975: MOV SS:[BP+8],AX
[L21] 976: RET 4
[BN4] 977: ;Exponenten zurückwandeln
[N22] 978: Z00C: XOR BL,80H
[W54] 979: ;Mantisse normalisieren
[4F2] 980: SI
[8C1] 981: JS Z00A
[KD1] 982: SHL DI,1
[CD1] 983: RCL CX,1
[XE1] 984: STI,1
[R31] 985: DEC BL
[NF1] 986: JNE Z009
[KS2] 987: ;Ergebnis runden
[VH2] 988: Z00A: ADD CX,128
[RC1] 989: ADC SI,0
[8G1] 990: JNC Z00B
[NF1] 991: RCR SI,1
[AE1] 992: RCR CX,1
[W41] 993: INC BL
[886] 994: ;Vorzeichen in das Ergebnis schreiben
[PF2] 995: Z00B: AND SI,AX
[7B3] 996: ;Ergebnis auf Stack
[R03] 997: MOV SS:[BP+6],CH
[R13] 998: MOV SS:[BP+7],SI
[413] 999: MOV SS:[BP+9],BL
[L21] 1000: RET 4
[000] 1001:

[L54] 1002: ;Zwei Singles dividieren
[RR2] 1003: DIVS: MOV BP,SP
[LP5] 1004: ;Eine der Zahlen 0, dann Ergebnis = 0
[FY3] 1005: CMP BYTE PTR SS:[BP+9],0
[L81] 1006: JE Z001
[ZY3] 1007: CMP BYTE PTR SS:[BP+5],0
[P81] 1008: JE Z001
[ZW2] 1009: ;Zahl 1 nach SI-BX
[W33] 1010: MOV SI,SS:[BP+6],CH
[S13] 1011: MOV BL,SS:[BP+8]
[RE1] 1012: MOV BH,0
[XU2] 1013: ;Zahl 2 nach DI-DX
[H13] 1014: MOV DI,SS:[BP+2]
[S13] 1015: MOV DL,SS:[BP+4]
[Z21] 1016: MOV DH,0
[KM6] 1017: ;Vorzeichen des Ergebnisses bestimmen
[XT1] 1018: MOV AL,BL
[XU1] 1019: XOR AL,DL
[IM1] 1020: OR AL,7FH
[F91] 1021: PUSH AX
[974] 1022: ;Mantissen ins Integerformat
[GH1] 1023: OR BL,80H
[SH1] 1024: OR DL,80H
```

```
[UY6] 1025: ;25 Bit berücksichtigen (24 + Rundungsbit)
[FN1] 1026: MOV CL,25
[D15] 1027: ;Ergebnis mit 0 initialisieren
[VU1] 1028: XOR BP,BP
[RE1] 1029: MOV AL,0
[925] 1030: ;Divident vom Divisor abziehen
[CF2] 1031: SBB BX,DX
[SS1] 1032: SBB BX,DX
[MG1] 1033: JNC Z00F
[M86] 1034: ;Unterlauf, dann Subtraktion rückgängig machen
[VR1] 1035: ADD SI,DI
[YR1] 1036: ADC BX,DX
[4A1] 1037: Z00F: CMC
[B76] 1038: ;Operationsbit ins Ergebnis schieben
[OD1] 1039: RCL BP,1
[SC1] 1040: RCL AL,1
[Z92] 1041: ;Divisor mal 2
[EF1] 1042: SHL SI,1
[8D1] 1043: RCL BX,1
[V26] 1044: ;Dividieren, bis Mantisse AL:BP voll
[V31] 1045: DEC CL
[NP1] 1046: AND AL,AL
[HK1] 1047: JNS Z00E
[5C4] 1048: ;Rundungsbit feststellen
[AU1] 1049: SUB SI,DI
[S81] 1050: SBB BX,DX
[TW0] 1051: CMC
[KS2] 1052: ;Ergebnis runden
[UA1] 1053: ADC BP,0
[KAL] 1054: ADC AL,0
[NG1] 1055: JNC Z00E
[9F5] 1056: ;Überlauf, dann eins nach rechts
[GD1] 1057: RCR AL,1
[SD1] 1058: RCR BP,1
[951] 1059: CLC
[342] 1060: Z00G: POP BX
[5V1] 1061: MOV DX,BP
[8W1] 1062: MOV BP,SP
[866] 1063: ;Vorzeichen in das Ergebnis schreiben
[UPI] 1064: AND AL,BL
[CR4] 1065: ;Ergebnismantisse auf Stack
[Z23] 1066: MOV SS:[BP+6],DX
[Z03] 1067: MOV SS:[BP+8],AL
[W13] 1068: ;Exponenten ins Integerformat
[P13] 1069: MOV AL,SS:[BP+9]
[A13] 1070: MOV AH,SS:[BP+5]
[3A5] 1071: ;Exponenten ins Integerformat
[W12] 1072: XOR SI,SI
[E55] 1073: ;Exponenten voneinander abziehen
[ZN1] 1074: ADD AL,CL
[SR1] 1075: SUB AL,AH
[Z23] 1076: ;Über-/Unterlauf, dann 0
[MC1] 1077: JO Z001
[SS4] 1078: ;Ergebnisexponent auf Stack
[9S1] 1079: XOR AX,80H
[Z13] 1080: MOV SS:[BP+9],AL
[MRI] 1081: Z00H: RET 4
[AF2] 1082: ;Ergebnis ist 0
[6K2] 1083: Z00I: XOR AX,AX
[X13] 1084: MOV SS:[BP+6],AX
[Z23] 1085: MOV SS:[BP+8],AX
[L21] 1086: RET 4
[000] 1087:

[H36] 1088: ;Natürlichen Logarithmus berechnen
[RH2] 1089: Z00J: MOV BP,SP
[HW4] 1090: ;Ist x = 0, dann 0 zurückgeben
[K13] 1091: MOV BL,SS:[BP+5]
[YP1] 1092: AND BL,BL
[TH1] 1093: JE Z00J0
[OW4] 1094: ;Ist x < 0, dann 0 zurückgeben
[UPI] 1095: MOV AL,128
[B13] 1096: TEST SS:[BP+4],AL
[XH1] 1097: JE Z00J1
[HW1] 1098: XOR AX,AX
[H13] 1099: MOV SS:[BP+2],AX
[R13] 1100: MOV SS:[BP+4],AX
[MD1] 1101: Z00J0: RET
[YK4] 1102: ;X zwischen 0.5 und 1 bringen
[V53] 1103: Z00J1: MOV AL,128
[NB6] 1104: ;X zwischen 1/SQR(2) und SQR(2) bringen
[EK4] 1105: CMP WORD PTR SS:[BP+4],8035H
[BH1] 1106: JNE Z00K
[FK4] 1107: CMP WORD PTR SS:[BP+2],04F4H
[MS2] 1108: Z00K: JAE Z00L
[NH3] 1109: INC BYTE PTR SS:[BP+5]
[R31] 1110: DEC BL
[X92] 1111: Z00L: PUSH BX
[UH1] 1112: ;X1 = X - 1
[NN2] 1113: PUSH SS:[BP+4]
[FN2] 1114: PUSH SS:[BP+2]
[F12] 1115: MOV AX,8180H
[P11] 1116: PUSH AX
[WV1] 1117: XOR AX,AX
[F91] 1118: PUSH AX
[EH1] 1119: CALL ADDS
[UH1] 1120: ;X2 = X + 1
[DL2] 1121: PUSH SS:[BP+16]
[SL2] 1122: PUSH SS:[BP+14]
[W12] 1123: MOV AX,8180H
[F91] 1124: PUSH AX
[WV1] 1125: XOR AX,AX
[F91] 1126: PUSH AX
[EH1] 1127: CALL ADDS
[P11] 1128: ;X1 = X1 / X2
[3L1] 1129: CALL IVS
[UW2] 1130: ;X = X1: X1 = X1 x X1
[023] 1131: MOV AX,SS:[BP+8]
[S13] 1132: MOV SS:[BP+16],AX
[F91] 1133: PUSH AX
[W13] 1134: MOV AX,SS:[BP+6]
[CS3] 1135: MOV SS:[BP+14],AX
[F91] 1136: PUSH AX
[ZL1] 1137: CALL MULS

[LU1] 1138: ;X2 = X1 x X1
[Q23] 1139: MOV AX,SS:[BP+8]
[393] 1140: MOV BX,SS:[BP+10]
[L91] 1141: PUSH BX
[F91] 1142: PUSH AX
[L91] 1143: PUSH BX
[F91] 1144: PUSH AX
[ZL1] 1145: CALL MULS
[TH2] 1146: ;X2 = X2 + .4342958
[Z52] 1147: MOV AX,7F5EH
[F91] 1148: PUSH AX
[Z32] 1149: MOV AX,574DH
[F91] 1150: PUSH AX
[EH1] 1151: CALL ADDS
[MU1] 1152: ;X2 = X2 x X1
[XK2] 1153: PUSH SS:[BP+12]
[NK2] 1154: PUSH SS:[BP+10]
[ZL1] 1155: CALL MULS
[NW2] 1156: ;X2 = X2 + .5765843
[A12] 1157: MOV AX,8013H
[F91] 1158: PUSH AX
[Y32] 1159: MOV AX,9B80H
[F91] 1160: PUSH AX
[EH1] 1161: CALL ADDS
[MU1] 1162: ;X2 = X2 x X1
[XK2] 1163: PUSH SS:[BP+12]
[NK2] 1164: PUSH SS:[BP+10]
[ZL1] 1165: CALL MULS
[1H2] 1166: ;X2 = X2 + .9618908
[P12] 1167: MOV AX,8076H
[F91] 1168: PUSH AX
[Z22] 1169: MOV AX,3893H
[F91] 1170: PUSH AX
[EH1] 1171: CALL ADDS
[HU1] 1172: ;X1 = X1 x X2
[ZL1] 1173: CALL MULS
```

```
[AB2] 1174: ;X1 = X1 + 2.88539
[Y12] 1175: MOV AX,8238H
[F91] 1176: PUSH AX
[MB2] 1177: MOV AX,0AA3BH
[F91] 1178: PUSH AX
[EH1] 1179: CALL ADDS
[DR1] 1180: ;X1 = X1 x X
[DL2] 1181: PUSH SS:[BP+16]
[SL2] 1182: PUSH SS:[BP+14]
[ZL1] 1183: CALL MULS
[KM7] 1184: ;Geretteten Exponenten zum Ergebnis addieren
[L83] 1185: MOV AL,SS:[BP+10]
[LE1] 1186: MOV AH,0
[XR1] 1187: XOR AL,80H
[SC1] 1188: JE Z00H
[YT1] 1189: MOV BL,AL
[9L1] 1190: JNS Z00H
[R41] 1191: NEG AL
[TH2] 1192: Z00H: MOV AH,80H
[MD2] 1193: Z00H: AND AL,AL
[KD1] 1194: JS Z000
[9D1] 1195: SHL AL,1
[F31] 1196: DEC AH
[XK1] 1197: JMP Z00H
[WK2] 1198: Z000: OR BL,7FH
[UPI] 1199: AND AL,BL
[Y92] 1200: Z00P: PUSH AX
[WV1] 1201: XOR AX,AX
[F91] 1202: PUSH AX
[EH1] 1203: CALL ADDS
[YC2] 1204: ;X = X x 1/LOG(2)
[S12] 1205: MOV AX,8031H
[F91] 1206: PUSH AX
[T12] 1207: MOV AX,7218H
[F91] 1208: PUSH AX
[ZL1] 1209: CALL MULS
[7B3] 1210: ;Ergebnis auf Stack
[ZG2] 1211: POP SS:[BP+14]
[7H2] 1212: POP SS:[BP+16]
[971] 1213: POP AX
[3P0] 1214: RET
[000] 1215:

[ZB4] 1216: ;Zwei Singles vergleichen
[YH2] 1217: Z00Q: MOV BP,SP
[B36] 1218: ;Vergleichsergebnisse initialisieren
[F91] 1219: MOV DI,0
[Z05] 1220: ;Zahl 1 nach AL:SI, Exponent in AH
[N33] 1221: MOV SI,SS:[BP+6]
[023] 1222: MOV AX,SS:[BP+8]
[DP5] 1223: ;Zahl 2 nach BL:DI, Exponent in BH
[H13] 1224: MOV DI,SS:[BP+2]
[W13] 1225: MOV BX,SS:[BP+4]
[KM4] 1226: ;Vorzeichen verschieden?
[ZU1] 1227: MOV AL,AL
[U11] 1228: MOV CH,BL
[AU1] 1229: XOR CH,CL
[7B1] 1230: JS Z00V
[Z74] 1231: ;Mantissen ins Integerformat
[CH1] 1232: OR AX,80H
[GH1] 1233: OR BL,80H
[YF3] 1234: ;Zahlen vergleichen
[U11] 1235: CMP AX,BX
[9K1] 1236: JNE Z00W
[AU1] 1237: CMP SI,DI
[U11] 1238: JE Z00U
[U22] 1239: Z00R: JA Z00S
[S81] 1240: XOR CL,80H
[WV3] 1241: ;Vorzeichen auswerten
[T62] 1242: Z00S: SHL CL,1
[U11] 1243: JC Z00W
[S81] 1244: ;Größen
[P12] 1245: Z00T: INC DL
[V91] 1246: ;Gleich
[SC1] 1247: Z00U: RET
[P62] 1248: Z00V: SHL AL,1
[YH1] 1249: JNC Z00T
[ED1] 1250: ;Kleiner
[Z12] 1251: Z00W: DEC DL
[ZP0] 1252: RET

[000] 1253:
[603] 1254: ;Datei BX schließen
[9A2] 1255: Z00X: PUSH AX
[PP2] 1256: MOV BL,DS:[BX]
[U04] 1257: ;Pseudo-Handle +SCRN: *?
[4L1] 1258: CMP BL,-1
[6D1] 1259: JE Z00Y
[HN3] 1260: ;MS-DOS-Systemdatei?
[SD1] 1261: CMP BL,5
[XC1] 1262: JB Z00Y
[WV1] 1263: XOR AX,AX
[LW1] 1264: MOV ES,AX
[DC4] 1265: ;Pseudo-Handle +COM1: *?
[8L1] 1266: CMP BL,-2
[P51] 1267: JNE Z00X1
[XK5] 1268: ;Interrupts in Controller sperren
[B22] 1269: MOV DX,03F9H
[9V1] 1270: OUT DX,AL
[6G1] 1271: ;Interrupt 4 sperren
[482] 1273: OR AL,00010000B
[AS1] 1274: OUT 21H,AL
[B66] 1275: ;Interruptvektor 4 wiederherstellen
[B03] 1276: MOV AX,CS:[0003-100]
[WV2] 1277: MOV ES:[12*4],AX
[XM1] 1278: MOV AX,CS:[0003-L00+2]
[Y92] 1279: MOV ES:[12*4+2],AX
[8E2] 1280: JMP SHORT Z00Y1
[EC4] 1281: ;Pseudo-Handle +COM2: *?
[ED2] 1282: Z00X1: CMP BL,-3
[K51] 1283: JNE Z00X0
[Z22] 1284: MOV DX,02F9H
[9V1] 1285: OUT DX,AL
[3G3] 1286: ;Interrupt 3 sperren
[6G1] 1287: IN AL,21H
[S82] 1288: OR AL,00001000B
[AS1] 1289: OUT 21H,AL
[DG3] 1290: MOV AX,CS:[0004-L00]
[UW2] 1291: MOV ES:[11*4],AX
[WV3] 1292: MOV AX,CS:[0004-L00+2]
[8E2] 1293: MOV ES:[11*4+2],AX
[GL2] 1294: JMP SHORT Z00Y
[B24] 1295: ;MS-DOS-Datei schließen
[V52] 1296: Z00X0: MOV AH,3EH
[NB1] 1297: IN AL,21H
[N42] 1298: Z00Y: POP BX
[ZP0] 1299: RET
[000] 1300:

[DP3] 1301: ;Exponentialfunktion
[B42] 1302: Z00Z: POP ES
[D62] 1303: ;X = X x 1/LOG(2)
[U12] 1304: MOV AX,8138H
[F91] 1305: PUSH AX
[H82] 1306: MOV AX,0AA3BH
[F91] 1307: PUSH AX
[ZL1] 1308: CALL MULS
[U87] 1309: ;X = X - INT(X), INT(X) als Exponent auf Stack
[HW1] 1310: XOR DX,DX
[WV1] 1311: XOR AX,AX
[Z23] 1312: MOV BX,SS:[BP+8]
[9N1] 1313: CMP BH,129
[461] 1314: JB A002
[TT1] 1315: MOV AH,BL
[8H1] 1316: OR AH,80H
[K13] 1317: MOV AL,SS:[BP+7]
[W13] 1318: MOV DL,SS:[BP+6]
[ZP2] 1319: A000: CMP BH,144
[KAL] 1320: JAE A001
[DE1] 1321: SHR AX,1
```



```

[UA1] 1322: ADC DH,0
[SA1] 1323: INC BH
[AC1] 1324: JMP A000
[AB2] 1325: A001: AND BL,BL
[MD1] 1326: JNS A002
[351] 1327: NEG AX
[571] 1328: AND DX,DX
[DE1] 1329: JZ A002
[231] 1330: DEC AX
[E72] 1331: A002: PUSH AX
[1P2] 1332: PUSH SS:[BP+8]
[VW2] 1333: PUSH SS:[BP+6]
[RE1] 1334: MOV BH,0
[LS1] 1335: AND AX,AX
[P61] 1336: JE A005
[RD1] 1337: JNS A003
[351] 1338: NEG AX
[6G2] 1339: A003: MOV BH,144
[RX1] 1340: A004: DEC BH
[ND2] 1341: SHL AX,1
[UD1] 1342: JNS A004
[6H2] 1343: A005: AND BL,80H
[SR1] 1344: AND AH,7FH
[EL1] 1345: OR AH,BL
[ET1] 1346: MOV BL,AH
[PT1] 1347: MOV AH,AL
[LS1] 1348: MOV AL,0
[151] 1349: XOR BL,80H
[LP1] 1350: PUSH BX
[F91] 1351: PUSH AX
[EH1] 1352: CALL ADDS
[RW2] 1353: ;X1 = X x 2.149876E-5
[F12] 1354: MOV AX,7134H
[F91] 1355: PUSH AX
[332] 1356: MOV AX,583CH
[F91] 1357: PUSH AX
[1P2] 1358: PUSH SS:[BP+8]
[VW2] 1359: PUSH SS:[BP+6]
[ZL1] 1360: CALL MULS
[MT2] 1361: ;X1 = X1 + 1.435231E-4
[X12] 1362: MOV AX,7416H
[F91] 1363: PUSH AX
[M42] 1364: MOV AX,7E77H
[F91] 1365: PUSH AX
[EH1] 1366: CALL ADDS
[DR1] 1367: ;X1 = X1 x X
[XK2] 1368: PUSH SS:[BP+12]
[NK2] 1369: PUSH SS:[BP+10]
[ZL1] 1370: CALL MULS
[002] 1371: ;X1 = X1 + 1.342263E-3
[A32] 1372: MOV AX,772FH
[F91] 1373: PUSH AX
[PD2] 1374: MOV AX,0EEDFH
[F91] 1375: PUSH AX
[DR1] 1376: CALL ADDS
[DR1] 1377: ;X1 = X1 x X
[XK2] 1378: PUSH SS:[BP+12]
[NK2] 1379: PUSH SS:[BP+10]
[ZL1] 1380: CALL MULS
[GT2] 1381: ;X1 = X1 + 9.14012E-3
[D42] 1382: MOV AX,7A1DH
[F91] 1383: PUSH AX
[Z22] 1384: MOV AX,8417H
[F91] 1385: PUSH AX
[EH1] 1386: CALL ADDS
[DR1] 1387: ;X1 = X1 x X
[XK2] 1388: PUSH SS:[BP+12]
[NK2] 1389: PUSH SS:[BP+10]
[ZL1] 1390: CALL MULS
[002] 1391: ;X1 = X1 + .5550512E-1
[S32] 1392: MOV AX,7C63H
[F91] 1393: PUSH AX
[N22] 1394: MOV AX,5959H
[F91] 1395: PUSH AX
[EH1] 1396: CALL ADDS
[DR1] 1397: ;X1 = X1 x X
[XK2] 1398: PUSH SS:[BP+12]
[NK2] 1399: PUSH SS:[BP+10]
[ZL1] 1400: CALL MULS
[4G2] 1401: ;X1 = X1 + .2402264
[542] 1402: MOV AX,7E75H
[F91] 1403: PUSH AX
[MC2] 1404: MOV AX,0FDE9H
[F91] 1405: PUSH AX
[EH1] 1406: CALL ADDS
[DR1] 1407: ;X1 = X1 x X
[XK2] 1408: PUSH SS:[BP+12]
[NK2] 1409: PUSH SS:[BP+10]
[ZL1] 1410: CALL MULS
[5H2] 1411: ;X1 = X1 + 6931472
[812] 1412: MOV AX,8031H
[F91] 1413: PUSH AX
[T12] 1414: MOV AX,7218H
[F91] 1415: PUSH AX
[EH1] 1416: CALL ADDS
[DR1] 1417: ;X1 = X1 x X
[XK2] 1418: PUSH SS:[BP+12]
[NK2] 1419: PUSH SS:[BP+10]
[ZL1] 1420: CALL MULS
[ZH1] 1421: ;X1 = X1 + 1
[712] 1422: MOV AX,8100H
[F91] 1423: PUSH AX
[WV1] 1424: XOR AX,AX
[F91] 1425: PUSH AX
[EH1] 1426: CALL ADDS
[NB6] 1427: ;Zwischenergebnisse vom Stack holen
[971] 1428: POP AX
[DT1] 1429: POP BX
[HT1] 1430: POP CX
[HT1] 1431: POP CX
[HT1] 1432: POP CX
[NL5] 1433: ;Gesicherten Exponenten addieren
[EN1] 1434: ADD BH,CL
[8C1] 1435: JNE A006
[TK3] 1436: ;Überlauf, Ergebnis 0
[WV1] 1437: XOR AX,AX
[3W1] 1438: XOR AX,BX
[783] 1439: ;Ergebnis auf Stack
[XK3] 1440: A006: MOV SS:[BP+16],AX
[Z53] 1441: MOV SS:[BP+18],BX
[U91] 1442: PUSH ES
[3P0] 1443: RET
[000] 1444:
[XK2] 1445: ;Integerfunktion
[VW2] 1446: A000: MOV BP,SP
[S13] 1447: MOV AX,SS:[BP+4]
[0Y4] 1448: ;Nachkommastellen vorhanden?
[AN1] 1449: CMP AH,152
[TB1] 1450: JAE J005
[SC2] 1451: ;Zahl negativ?
[RP1] 1452: AND AL,AL
[EH1] 1453: JNS J001
[734] 1454: ;Ja, dann .9999999 abziehen
[NW2] 1455: PUSH SS:[BP+2]
[FN2] 1456: PUSH SS:[BP+2]
[S32] 1457: MOV AX,80FFH
[F91] 1458: PUSH AX
[DD2] 1459: MOV AX,0FFFFH
[F91] 1460: PUSH AX
[EH1] 1461: CALL ADDS
[8W1] 1462: MOV BP,SP
[AB1] 1463: ADD BP,4
[ZL2] 1464: POP SS:[BP+2]
[FL2] 1465: POP SS:[BP+4]
[K74] 1466: ;AND-Maske initialisieren
[R33] 1467: J001: MOV AX,0FFFFH
[9Y1] 1468: MOV BL,0
[NW4] 1469: ;Exponent zwischen 144 und 1517
[6R1] 1470: MOV CL,144
[403] 1471: SUB CL,SS:[BP+5]

```

```

[NB1] 1472: JAE J002
[PD3] 1473: ;Ja, dann verschieben
[PB1] 1474: ADD CL,8
[PS1] 1475: SHL AL,CL
[9H2] 1476: JMP SHORT J004
[M52] 1477: J002: MOV AL,0
[0K4] 1478: ;Exponent zwischen 136 und 1447
[5E1] 1479: SUB CL,8
[SB1] 1480: JAE J003
[PB1] 1481: ADD CL,8
[BS1] 1482: SHL AH,CL
[9H2] 1483: JMP SHORT J004
[552] 1484: J003: MOV AH,0
[5X4] 1485: ;Exponent zwischen 128 und 1357
[5E1] 1486: CMP CL,8
[1C1] 1487: JAE J006
[LS1] 1488: SHL BL,CL
[584] 1489: ;X mit Maske verknüpfen
[NL3] 1490: J004: AND SS:[BP+2],AX
[XK2] 1491: AND SS:[BP+4],BL
[3A1] 1492: RET
[AF2] 1493: ;Ergebnis ist 0
[VN3] 1494: J006: MOV SS:[BP+2],AX
[R13] 1495: MOV SS:[BP+4],AX
[3P0] 1496: RET
[000] 1497:
[HF1] 1498: ;Cosinus
[P12] 1499: A007: POP ES
[N51] 1500: ;X = X + PI/2
[X12] 1501: MOV AX,8149H
[F91] 1502: PUSH AX
[T42] 1503: MOV AX,0FDAH
[F91] 1504: PUSH AX
[EH1] 1505: CALL ADDS
[U91] 1506: PUSH ES
[000] 1507:
[441] 1508: ;Sinus
[R12] 1509: A008: POP ES
[ZC3] 1510: ;X = X * (2 x PI))
[U32] 1511: MOV AX,7E22H
[F91] 1512: PUSH AX
[RA2] 1513: MOV AX,0F981H
[F91] 1514: PUSH AX
[ZL1] 1515: CALL MULS
[8W1] 1516: MOV BP,SP
[FN2] 1517: PUSH SS:[BP+2]
[9B2] 1518: PUSH SS:[BP]
[9D1] 1519: CALL J000
[8W1] 1520: MOV BP,SP
[UX3] 1521: CMP BYTE PTR SS:[BP+3],0
[7T1] 1522: JZ A009
[E84] 1523: XOR BYTE PTR SS:[BP+2],80H
[NA2] 1524: A009: CALL ADDS
[8W1] 1525: MOV BP,SP
[UX3] 1526: CMP BYTE PTR SS:[BP+3],0
[U71] 1527: JZ A00A
[E84] 1528: XOR BYTE PTR SS:[BP+2],80H
[NF1] 1529: ;X = X + .25
[3U2] 1530: A00A: MOV AX,7F00H
[F91] 1531: PUSH AX
[WV1] 1532: XOR AX,AX
[F91] 1533: PUSH AX
[EH1] 1534: CALL ADDS
[8W1] 1535: MOV BP,SP
[M31] 1536: ;X = 0 ?
[0Y4] 1537: TEST BYTE PTR SS:[BP+2],80H
[X71] 1538: JE A00B
[GB1] 1539: ;X = X * 5
[312] 1540: MOV AX,8000H
[F91] 1541: PUSH AX
[WV1] 1542: XOR AX,AX
[F91] 1543: PUSH AX
[EH1] 1544: CALL ADDS
[8W1] 1545: MOV BP,SP
[U71] 1546: ;X = ABS (X)
[U74] 1547: AND BYTE PTR SS:[BP+2],7FH
[RP1] 1548: ;X = X - .25
[942] 1549: MOV AX,7F80H
[F91] 1550: PUSH AX
[WV1] 1551: XOR AX,AX
[F91] 1552: PUSH AX
[9M2] 1553: JMP SHORT A00D
[0M4] 1554: A00B: CMP BYTE PTR SS:[BP+3],0
[U01] 1555: JE A00C
[U01] 1556: ;X = - X
[E84] 1557: XOR BYTE PTR SS:[BP+2],80H
[NF1] 1558: ;X = X + .25
[SU2] 1559: A00C: MOV AX,7F00H
[F91] 1560: PUSH AX
[WV1] 1561: XOR AX,AX
[F91] 1562: PUSH AX
[0B2] 1563: A00D: CALL ADDS
[8W1] 1564: MOV BP,SP
[CT1] 1565: ;X1 = X x X
[FN2] 1566: PUSH SS:[BP+2]
[9B2] 1567: PUSH SS:[BP]
[FN2] 1568: PUSH SS:[BP+2]
[9B2] 1569: PUSH SS:[BP]
[ZL1] 1570: CALL MULS
[E82] 1571: ;X2 = X1 x -14.38139
[M22] 1572: MOV AX,84E6H
[F91] 1573: PUSH AX
[W32] 1574: MOV AX,1A2DH
[F91] 1575: PUSH AX
[8W1] 1576: MOV BP,SP
[VN2] 1577: PUSH SS:[BP+6]
[VN2] 1578: PUSH SS:[BP+4]
[ZL1] 1579: CALL MULS
[VN2] 1580: ;X2 = X2 + 42.0078
[D22] 1581: MOV AX,8628H
[F91] 1582: PUSH AX
[132] 1583: MOV AX,07FBH
[F91] 1584: PUSH AX
[EH1] 1585: CALL ADDS
[MU1] 1586: ;X2 = X2 x X1
[8W1] 1587: MOV BP,SP
[VN2] 1588: PUSH SS:[BP+6]
[NN2] 1589: PUSH SS:[BP+4]
[ZL1] 1590: CALL MULS
[9H2] 1591: ;X2 = X2 - 76.79417
[U22] 1592: MOV AX,8799H
[F91] 1593: PUSH AX
[S22] 1594: MOV AX,6899H
[F91] 1595: PUSH AX
[EH1] 1596: CALL ADDS
[MU1] 1597: ;X2 = X2 x X1
[8W1] 1598: MOV BP,SP
[VN2] 1599: PUSH SS:[BP+6]
[NN2] 1600: PUSH SS:[BP+4]
[ZL1] 1601: CALL MULS
[7G2] 1602: ;X2 = X2 + 81.60522
[722] 1603: MOV AX,8723H
[F91] 1604: PUSH AX
[832] 1605: MOV AX,35DFH
[F91] 1606: PUSH AX
[EH1] 1607: CALL ADDS
[MU1] 1608: ;X2 = X2 x X1
[8W1] 1609: MOV BP,SP
[VN2] 1610: PUSH SS:[BP+6]
[NN2] 1611: PUSH SS:[BP+4]
[ZL1] 1612: CALL MULS
[ZL2] 1613: ;X2 = X2 - 41.3147
[P22] 1614: MOV AX,86A5H
[F91] 1615: PUSH AX
[Z42] 1616: MOV AX,8DE7H
[F91] 1617: PUSH AX
[EH1] 1618: CALL ADDS
[MU1] 1619: ;X2 = X2 x X1
[8W1] 1620: MOV BP,SP
[VN2] 1621: PUSH SS:[BP+6]

```

```

[NN2] 1622: PUSH SS:[BP+4]
[ZL1] 1623: CALL MULS
[G22] 1624: ;X2 = X2 + 6.382
[S22] 1625: MOV AX,8349H
[F91] 1626: PUSH AX
[T42] 1627: MOV AX,0FDAH
[F91] 1628: PUSH AX
[EH1] 1629: CALL ADDS
[CS1] 1630: ;X = X2 x X1
[8W1] 1631: MOV BP,SP
[FL2] 1632: POP SS:[BP+4]
[NL2] 1633: POP SS:[BP+6]
[ZL1] 1634: CALL MULS
[YU6] 1635: ;Gerettete Rückspringadresse auf Stack
[U91] 1636: PUSH ES
[3P0] 1637: RET
[000] 1638:
[AU8] 1639: ;Programm beenden (*END* oder letzte Zeile erreicht)
[6E2] 1640: LBI: MOV BX,3
[184] 1641: ;Alle Dateien schließen
[F82] 1642: X010: CALL Z00X
[341] 1643: DEC BX
[M21] 1644: JNE X010
[465] 1645: ;Cursorform wiederherstellen
[7R2] 1646: MOV CX,DS:[15]
[EP1] 1647: MOV AH,01H
[G81] 1648: INT 10H
[9V0] 1649: ;Raus
[L71] 1650: MOV AH,4CH
[N81] 1651: INT 21H
[000] 1652:
[Z71] 1653: ;Zwei Integerwerte miteinander vergleichen
[CE2] 1654: IVGL MACRO CMD
[YF1] 1655: LOCAL LBL
[8W1] 1656: MOV BP,SP
[U74] 1657: ;Ergebnis initialisieren
[NW1] 1658: XOR DX,DX
[YF3] 1659: ;Zahlen vergleichen
[S13] 1660: MOV AX,SS:[BP+4]
[293] 1661: CMP AX,SS:[BP+2]
[465] 1662: ;Nicht erfolgreich, dann überspringen
[7A1] 1663: CMD LBL
[344] 1664: ;Ergebnis = -1 (*true*)
[B41] 1665: DEC DX
[783] 1666: ;Ergebnis auf Stack
[Y73] 1667: MOV SS:[BP+4],DX
[B21] 1668: RET 2
[NP0] 1669: ENDM
[000] 1670:
[NW2] 1671: ;Integer >=
[182] 1672: L02: IVGL JNG
[000] 1673:
[DC2] 1674: ;Integer <=
[A82] 1675: L03: IVGL JNL
[000] 1676:
[XC2] 1677: ;Integer ==
[782] 1678: L04: IVGL JNE
[000] 1679:
[0M2] 1680: ;Integer >=
[RC2] 1681: L05: IVGL JNGE
[000] 1682:
[S12] 1683: ;Integer <=
[002] 1684: L06: IVGL JNLE
[000] 1685:
[TL2] 1686: ;Integer <>
[A52] 1687: L07: IVGL JE
[000] 1688:
[XK7] 1689: ;Operation zwischen zwei Integer ausführen
[BK2] 1690: INTOP MACRO CMD
[8W1] 1691: MOV BP,SP
[M13] 1692: MOV AX,SS:[BP+2]
[N72] 1693: CMD SS:[BP+4],AX
[B21] 1694: RET 2
[NP0] 1695: ENDM
[000] 1696:
[9A2] 1697: ;Integer ==
[692] 1698: L08: INTOP ADD
[000] 1699:
[HA2] 1700: ;Integer >= DX,DX
[202] 1701: L09: INTOP SUB
[000] 1702:
[LY2] 1703: ;Integer <=
[K82] 1704: L0A: INTOP AND
[000] 1705:
[TP2] 1706: ;Integer <=
[CB2] 1707: L0B: INTOP OR
[000] 1708:
[CB2] 1709: ;Integer <=
[HE2] 1710: L0C: INTOP XOR
[000] 1711:
[XF2] 1712: ;Integer >=
[7B2] 1713: L10: MOV BP,SP
[S13] 1714: MOV AX,SS:[BP+4]
[DT3] 1715: MUL WORD PTR SS:[BP+2]
[R13] 1716: MOV SS:[BP+4],AX
[B21] 1717: RET 2
[000] 1718:
[R15] 1719: ;Zwei Integerwerte dividieren
[NK2] 1720: INTDV MACRO REG
[WK1] 1721: MOV BP,SP
[S13] 1722: MOV AX,SS:[BP+4]
[8W1] 1723: XOR DX,DX
[W83] 1724: IDIV WORD PTR SS:[BP+2]
[W93] 1725: MOV SS:[BP+4],REG
[B21] 1726: RET 2
[NP0] 1727: ENDM
[000] 1728:
[DG2] 1729: ;Integer <=
[0A2] 1730: L0E: INTDV AX
[000] 1731:
[493] 1732: ;Integer <=
[DA2] 1733: L0F: INTDV DX
[000] 1734:
[813] 1735: ;Integer <=
[U82] 1736: L10: MOV BP,SP
[X73] 1737: NOT WORD PTR SS:[BP+2]
[3P0] 1738: RET
[000] 1739:
[NP5] 1740: ;Integer negatives Vorzeichen
[YE2] 1741: L11: MOV BP,SP
[BR3] 1742: NEG WORD PTR SS:[BP+2]
[3P0] 1743: RET
[000] 1744:
[867] 1745: ;Einen String von DS:SI nach ES:DI kopieren
[SX1] 1746: TRANS MACRO
[U01] 1747: CLD
[CV0] 1748: LODSB
[NK0] 1749: STOSB
[ZU1] 1750: MOV CL,AL
[V81] 1751: MOV CH,0
[3X1] 1752: REP MOVSB
[000] 1753: ENDM
[000] 1754:
[CM5] 1755: ;Stringtransfer Speicher -> Stapel
[ZP2] 1756: L12: MOV BP,SP
[4X4] 1757: ;Rückspringadresse sichern
[FR2] 1758: MOV BX,SS:[BP]
[LL3] 1759: ;Datensegment sichern
[RV1] 1760: MOV DX,DS
[M05] 1761: ;Adresse des Strings vom Stapel
[Y13] 1762: L13: MOV SI,SS:[BP+2]
[7B5] 1763: ;256 Bytes auf Stapel reservieren
[V81] 1764: SUB SP,250
[UG3] 1765: ;Zieladresse setzen

```

Listing 2. Der Quelltext der SBasic-Laufzeitbibliothek »sbasic.asm«


```
[9W1] 1766: MOV DI,SP
[SW1] 1767: MOV AX,SS
[1W1] 1768: MOV ES,AX
[6C3] 1769: ;String übertragen
[PW0] 1770: TRANS
[9A5] 1771: ;Datenelement wiederherstellen
[6W1] 1772: MOV DS,DX
[6V1] 1773: JMP BX
[000] 1774:
[MV5] 1775: ;Stringtransfer Stapel -> Speicher
[XF2] 1776: L13: MOV SI,SP
[K46] 1777: ;Adresse des Quellstrings nach DS:SI
[1D1] 1778: ADD SI,2
[P91] 1779: PUSH DS
[SW1] 1780: MOV AX,SS
[XV1] 1781: MOV DS,AX
[B96] 1782: ;Adresse des Zielstrings vom Stapel
[6U1] 1783: XOR DI,DI
[L83] 1784: MOV ES,SS:[SI+256]
[DL1] 1785: ;Kopieren TRANS
[G71] 1787: POP DS
[UV4] 1788: ;String vom Stack entfernen
[G71] 1789: RET 256
[000] 1790:
[ISA] 1791: ;PRINT CRLF (Abschluss einer PRINT-Anweisung
ohne Strichpunkt)
[AF2] 1792: L14: MOV BP,SP
[UV3] 1793: ;Dateinummer vom Stack
[S13] 1794: MOV BX,SS:[BP+2]
[VN2] 1795: ;CHRS(13) ausgeben
[AT7] 1796: MOV AL,0DH
[SA7] 1797: ;Relative Adressierung hier nicht möglich !
!
[AG2] 1798: MOV DX,X006-L00
[D61] 1799: CALL DX
[134] 1800: ;Ausgabe auf Bildschirm?
[ZV3] 1801: CMP PTR DS:[BX],-1
[C91] 1802: JE X140
[7N8] 1803: ;Nein, dann noch ein 'Line Feed' hinterhers
chicken
[UM1] 1804: MOV AL,10
[AG2] 1805: MOV DX,X006-L00
[D61] 1806: CALL DX
[FB1] 1807: X140: RET
[000] 1808:
[283] 1809: ;PRINT String
[EF2] 1810: L15: MOV BP,SP
[481] 1811: ADD BP,2
[743] 1812: ;Dateinummer holen
[N73] 1813: MOV BX,SS:[BP+256]
[LH3] 1814: ;Stringlänge nach CX
[7R2] 1815: MOV CL,SS:[BP]
[VE1] 1816: MOV CH,0
[5K1] 1817: JNC X151
[F12] 1818: INC BP
[0D2] 1819: ;Zeichen lesen
[EP2] 1820: MOV AL,SS:[BP]
[R91] 1821: PUSH CX
[L72] 1822: ;und ausgeben
[AG2] 1823: MOV DX,X006-L00
[D61] 1824: CALL DX
[W71] 1825: POP CX
[WH1] 1826: LOOP X150
[MZ1] 1827: RET 256
[000] 1828:
[257] 1829: ;Konvertierung Integer -> String (*STRS*)
[GH2] 1830: L16: SUB SP,252
[ZG4] 1831: ;Platz für String schaffen
[8W1] 1832: MOV BP,SP
[PD2] 1833: ;Integer holen
[973] 1834: MOV AX,SS:[BP+254]
[LK1] 1835: MOV BL,-
[SC2] 1836: ;Zahl negativ?
[L51] 1837: AND AX,AX
[RG1] 1838: JNS X160
[351] 1839: AX NEG AX
[7M1] 1840: MOV BL,-
[H74] 1841: ;Erstes Zeichen sichern
[M33] 1842: X160: MOV SS:[BP+1],BL
[NE1] 1843: ;Divisor
[AN1] 1844: MOV BX,10
[9T2] 1845: ;Ziffernzähler
[AN1] 1846: XOR CX,CX
[UX2] 1847: ;Division durch 10
[NK2] 1848: X161: XOR DX,DX
[V61] 1849: DIV BX
[PF3] 1850: ;Rest im ASCII-Format
[GE1] 1851: OR DL,0
[4S1] 1852: ;auf Stack
[V91] 1853: PUSH DX
[BD4] 1854: ;Ergebnis < 0, dann weiter
[L51] 1855: AND AX,AX
[WS1] 1856: MOV ES,AX
[B51] 1857: NEG CX
[4U1] 1858: MOV BL,CL
[6F1] 1859: MOV DI,2
[L78] 1860: ;Ziffern vom Stack holen und im String spei
chern
[242] 1861: X162: POP AX
[983] 1862: MOV SS:[BP+DI],AL
[L51] 1863: INC DI
[LK1] 1864: LOOP X162
[266] 1865: ;Mit " " abschließen, für *PRINT*
[Y44] 1866: MOV BYTE PTR SS:[BP+DI], "
[GK4] 1867: ;Länge des Strings setzen
[L11] 1868: INC BL
[DN2] 1869: MOV SS:[BP],BL
[3N2] 1870: JMP SS:[BP+252]
[000] 1871:
[LX8] 1872: ;CLSE, *CHRS(12)* in Datei 0 (Bildschirm)
schreiben
[BG2] 1873: L17: XOR BX,BX
[ZN1] 1874: MOV AL,12
[AG2] 1875: MOV DX,X006-L00
[371] 1876: JMP DX
[000] 1877:
[0U3] 1878: ;Zwei Strings addieren
[RG2] 1879: L18: MOV SI,SP
[Z03] 1880: ;DS:SI auf String 2
[1D1] 1881: ADD SI,2
[P91] 1882: PUSH DS
[SW1] 1883: MOV AX,SS
[XV1] 1884: MOV DS,AX
[H25] 1885: ;ES:DI auf das Ende von String 1
[SW1] 1886: MOV ES,AX
[R63] 1887: MOV CL,DS:[SI+256]
[YE1] 1888: MOV CH,1
[TV1] 1889: MOV DI,CX
[G93] 1890: ;Länge von String 1
[CV0] 1891: LODSB
[CR1] 1892: ADD DI,SI
[447] 1893: ;Anzahl der zu kopierenden Bytes berechnen
[L11] 1894: INC BL
[B51] 1895: XOR CX,CX
[TL1] 1896: ADD CX,255
[ES1] 1897: CMP AL,CL
[WD1] 1898: JAE X180
[ZU1] 1899: MOV DI,AL
[ZN5] 1900: ;Länge von String 1 aktualisieren
[H23] 1901: X180: ADD DS:[SI+255],CL
[713] 1902: ;Zeichen anhängen
[3X1] 1903: REP MOVSB
[G71] 1904: POP DS
[RD3] 1905: ;String 2 vergessen
[CT1] 1906: RET 256
[000] 1907:
[TR0] 1908: ;CHRS
[WH2] 1909: L19: SUB SP,252
[ZG4] 1910: ;Platz für String schaffen
```

```
[8W1] 1911: MOV BP,SP
[W42] 1912: ;Stringlänge
[UE1] 1913: MOV AL,1
[1D1] 1914: ;Zeichen
[763] 1915: MOV AH,SS:[BP+254]
[NM2] 1916: MOV SS:[BP],AX
[3N2] 1917: JMP SS:[BP+252]
[000] 1918:
[S22] 1919: ;SGN Integer
[YG2] 1920: L1A: MOV BP,SP
[U74] 1921: ;Ergebnis initialisieren
[HW1] 1922: XOR DX,DX
[WV3] 1923: ;Zahl mit 0 vergleichen
[503] 1924: CMP SS:[BP+2],DX
[9A1] 1925: JE X1A1
[CA1] 1926: JC X1A0
[B41] 1927: DEC DX
[BW2] 1928: JMP SHORT X1A1
[542] 1929: X1A0: INC DX
[763] 1930: ;Ergebnis auf Stack
[Y53] 1931: X1A1: MOV SS:[BP+2],DX
[3P0] 1932: RET
[000] 1933:
[U22] 1934: ;LOCATE x,y
[ZH2] 1935: L1B: MOV BP,SP
[L25] 1936: ;Aktuelle Bildschirmseite holen
[L71] 1937: MOV AH,0FH
[G81] 1938: INT 10H
[G83] 1939: ;Parameter vom Stack
[M13] 1940: MOV DL,SS:[BP+2]
[M13] 1941: MOV DH,SS:[BP+4]
[772] 1942: ;Um 1 verringern
[BZ1] 1943: SUB DX,0101H
[TW3] 1944: ;Cursor positionieren
[3R1] 1945: MOV AH,02H
[G81] 1946: INT 10H
[K53] 1947: ;Returnflag löschen
[H33] 1948: MOV BYTE PTR DS:[10],0
[L21] 1949: RET 4
[000] 1950:
[TU1] 1951: ;COLOR
[6H2] 1952: L1C: MOV BP,SP
[ZH2] 1953: ;Parameter holen
[C13] 1954: MOV AL,SS:[BP+4]
[413] 1955: MOV AH,SS:[BP+2]
[S83] 1956: ;Attribut berechnen
[BP1] 1957: MOV
[BS1] 1958: SHL AH,CL
[DM1] 1959: AND AH,70H
[YT1] 1960: MOV BL,AL
[MD1] 1961: AND AL,0FH
[ZK1] 1962: OR AL,AH
[8F1] 1963: MOV CL,3
[LS1] 1964: SHL BL,CL
[SH1] 1965: AND BL,09H
[KL1] 1966: OR BL,SL
[PX2] 1967: ;Attribut sichern
[LQ2] 1968: MOV DS:[9],AL
[L21] 1969: RET 4
[000] 1970:
[U11] 1971: ;WIDTH
[3P2] 1972: L1D: MOV AH,0FH
[G81] 1973: INT 10H
[4W3] 1974: ;CursorPosition holen
[7R1] 1975: MOV AH,03H
[G81] 1976: INT 10H
[8W1] 1977: MOV BP,SP
[TW2] 1978: ;Parameter lesen
[C13] 1979: MOV BL,SS:[BP+2]
[NL1] 1980: CMP BL,40
[4G1] 1981: JNE X1D3
[7R4] 1982: ;WIDTH 40* bereits an?
[DL1] 1983: CMP AH,40
[KAL] 1984: JE X1D2
[YA4] 1985: ;Grafik, dann Videomodus 5
[RD1] 1986: CMP AL,6
[CA1] 1987: JC X1D0
[6W0] 1988: ;Text
[UE1] 1989: MOV AL,1
[R42] 1990: JMP SHORT X1D1
[D92] 1991: X1D0: MOV AL,5
[9A4] 1992: ;Bildschirmbreite sichern
[S22] 1993: X1D1: DEC BL
[EG2] 1994: MOV DS:[11],BL
[QB3] 1995: ;Videomodus setzen
[VP1] 1996: MOV AH,09H
[G81] 1997: INT 10H
[8P2] 1998: X1D2: MOV AH,01H
[G81] 1999: INT 10H
[B21] 2000: RET 2
[4N2] 2001: ;WIDTH 80
[9D2] 2002: X1D3: CMP BL,80
[1G1] 2003: JNE X1D2
[E15] 2004: ;Bereite 80 Zeichen pro Zeile?
[TL1] 2005: CMP AH,80
[KAL] 2006: JE X1D2
[SE3] 2007: ;Grafikmodus aktiv?
[MD1] 2008: CMP AL,4
[PF1] 2009: JAE X1D4
[6W0] 2010: ;Text
[8F1] 2011: MOV AL,3
[7H1] 2012: JMP X1D1
[L92] 2013: X1D4: MOV AL,6
[7H1] 2014: JMP X1D1
[000] 2015:
[AR2] 2016: ;VIEW PRINT
[H22] 2017: L1E: MOV BP,SP
[SAS] 2018: ;Obere Grenze holen und sichern
[C13] 2019: MOV AL,SS:[BP+4]
[L31] 2020: DEC AL
[B62] 2021: MOV DS:[12],AL
[U55] 2022: ;Untere Grenze holen und sichern
[B13] 2023: MOV AL,SS:[BP+2]
[L31] 2024: DEC AL
[CG2] 2025: MOV DS:[13],AL
[U48] 2026: ;Cursor Home in Datei 0 (*SCRN*) schreib
en
[3W1] 2027: XOR BX,BX
[AG2] 2028: MOV AL,11
[AG2] 2029: MOV DX,X006-L00
[D61] 2030: CALL DX
[L21] 2031: RET 4
[000] 2032:
[V12] 2033: ;OPEN
[9K2] 2034: L1F: MOV SI,SP
[S23] 2035: ;DS:SI auf Namensstring
[LD1] 2036: ADD SI,8
[MV1] 2037: MOV BX,SI
[P91] 2038: PUSH DS
[SW1] 2039: MOV AX,SS
[XV1] 2040: MOV DS,AX
[X91] 2041: ;Länge
[CV0] 2042: LODSB
[LE1] 2043: MOV AH,0
[WV1] 2044: MOV DX,SI
[771] 2045: ;String mit 0 abschließen (MS-DOS-Format)
[ET1] 2046: ADD SI,AX
[6N2] 2047: MOV DS:[SI],AH
[R55] 2048: ;Ist es ein reserviertes Gerät?
[354] 2049: CMP BYTE PTR DS:[BX+5], "
[ZN1] 2050: JNE X1F00
[XH5] 2051: ;Gerätenamen in Großbuchstaben
[S13] 2052: MOV AX,DS:[BX+1]
[G22] 2053: AND AX,0DFDH
[H13] 2054: MOV CX,DS:[BX+3]
[S23] 2055: AND CX,0DFDH
[MX1] 2056: ;*SCRN*?
[AT1] 2057: MOV AX,"CS"
[XU1] 2058: JNE X1F02
[701] 2059: CMP CX,"NR"
[MG1] 2060: JE X1F01
```

```
[7K2] 2061: X1F00: JMP X1F0X
[WW3] 2062: ;Pseudo-Handle *SCRN*
[MP2] 2063: X1F01: MOV AL,-1
[SH1] 2064: JMP X1F1
[CV1] 2065: ;KYBD*?
[PX2] 2066: X1F02: CMP AX,"YK"
[BP1] 2067: JNE X1F03
[S71] 2068: CMP CX,"DB"
[MG1] 2069: JE X1F0
[YR2] 2070: JMP SHORT X1F00
[684] 2071: ;COM1* oder *COM2*?
[U22] 2072: X1F03: CMP AX,"OC"
[ZM1] 2073: JNE X1F00
[815] 2074: ;DH enthält Adresse der SIO
[L81] 2075: MOV DH,03H
[X44] 2076: CMP CX,("1" AND 0DFH) SHL 8 OR "M"
[9H1] 2077: JE X1F04
[V31] 2078: DEC DH
[DW4] 2079: CMP CX,("2" AND 0DFH) SHL 8 OR "M"
[ZM1] 2080: JNE X1F00
[SAS] 2081: ;Zugriff auf Baudratenregister
[SW2] 2082: X1F04: MOV DL,0FBH
[LE2] 2083: MOV AL,1000011B
[9V1] 2084: OUT DX,AL
[684] 2085: ;Baudrate aus String holen
[NF2] 2086: LEA [BX+6]
[V91] 2087: PUSH DX
[AW1] 2088: XOR CX,CX
[AC4] 2089: ;Nächstes Zeichen lesen
[7V1] 2090: X1F05: LODSB
[8F3] 2091: ;Ist es eine Ziffer?
[XK1] 2092: SUB AL,"0"
[Z01] 2093: JB X1F06
[SU1] 2094: CMP AL,10
[GN1] 2095: JAE X1F06
[YF3] 2096: ;Vorheriger Wert mal 10
[LE1] 2097: MOV AH,0
[LV1] 2098: MOV DI,AX
[UM1] 2099: MOV AL,10
[D71] 2100: MUL CX
[9G3] 2101: ;Neue Ziffer addieren
[ZP1] 2102: ADD AX,DI
[VY1] 2103: MOV CX,AX
[W81] 2104: JMP X1F05
[G25] 2105: ;Baudrate = 0, dann mit 300 vorbelegen
[HM2] 2106: X1F06: AND CX,CX
[V91] 2107: JNE X1F07
[4R1] 2108: MOV CX,300
[B35] 2109: ;Registerwert = 115200 / Baudrate
[CY2] 2110: X1F07: MOV AX,49664
[KF1] 2111: MOV CX,1
[Z61] 2112: DIV CX
[N71] 2113: POP DX
[L22] 2114: ;Baudrate ausgeben
[3Y1] 2115: MOV DL,0F8H
[9V1] 2116: OUT DX,AL
[G51] 2117: INC DX
[AT1] 2118: MOV AL,AH
[9V1] 2119: OUT DX,AL
[G51] 2120: INC DX
[G51] 2121: INC DX
[355] 2122: ;No Parity, 8 Databit, 1 Stopbit
[G22] 2123: MOV AL,00000011B
[9V1] 2124: OUT DX,AL
[G51] 2125: INC DX
[EM7] 2126: ;DTR*, *RTS* und *Interrupt Enable*
[LE2] 2127: MOV AL,0000011B
[9V1] 2128: OUT DX,AL
[SAS] 2129: ;Enable Receive-Data-Interrupt
[4Y1] 2130: MOV DL,0F8H
[ZP2] 2131: MOV AL,0000001B
[9V1] 2132: OUT DX,AL
[YG6] 2133: ;Dummy-Lesen zur Blockadeverhinderung
[B41] 2134: DEC DX
[GMI] 2135: IN AL,DX
[9L4] 2136: ;Interruptvektoren setzen
[P91] 2137: PUSH DS
[WV1] 2138: XOR AX,AX
[XV1] 2139: MOV DS,AX
[YM1] 2140: MOV CL,-2
[PD1] 2141: CMP DH,3
[EPI] 2142: JNC X1F08
[Z44] 2143: ;COM1* = Interrupt 12
[HK4] 2144: MOV WORD PTR CS:[U007-L00],AX
[4R3] 2145: MOV AX,OFFSETOF U009-L00
[6U2] 2146: XCHG AX,DS:[12*4]
[UG3] 2147: MOV CS:[U004-L00],AX
[8V1] 2148: MOV AX,CS
[FY2] 2149: XCHG AX,DS:[12*4+2]
[BL3] 2150: MOV CS:[U003-L00+2],AX
[GMI] 2151: ;Interrupt einschalten
[6G1] 2152: IN AL,21H
[W82] 2153: AND AL,1110111B
[G22] 2154: JMP SHORT X1F09
[Z44] 2155: ;COM2* = Interrupt 11
[T42] 2156: X1F08: DEC CL
[TK4] 2157: MOV WORD PTR CS:[U007-L00],AX
[ER3] 2158: MOV AX,OFFSETOF U00C-L00
[WT2] 2159: XCHG AX,DS:[11*4]
[XG3] 2160: MOV CS:[U004-L00],AX
[8V1] 2161: MOV AX,CS
[BY2] 2162: XCHG AX,DS:[11*4+2]
[BL3] 2163: MOV CS:[U004-L00+2],AX
[6G1] 2164: IN AL,21H
[V82] 2165: AND AL,1110111B
[DR2] 2166: X1F09: OUT 21H,AL
[G71] 2167: POP DS
[V46] 2168: ;Erneute Dummy-Lesen zur Sicherheit
[GMI] 2169: IN AL,DX
[9U1] 2170: MOV AL,CL
[ZM2] 2171: JMP SHORT X1F1
[BC5] 2172: ;Normale Datei Betriebsart holen
[SU3] 2173: X1F0X: MOV AL,DS:[BX-2]
[8D1] 2174: CMP AL,1
[8E1] 2175: JE X1F0
[ED1] 2176: CMP AL,3
[PF1] 2177: JNE X1F0
[UE1] 2178: MOV AL,1
[S14] 2179: ;Für Input oder Append
[DS2] 2180: X1F0: MOV AH,3DH
[N81] 2181: INT 21H
[UF1] 2182: JNC X1F1
[XK3] 2183: CMP BYTE PTR DS:[BX-2],0
[ZAL] 2184: JE X1F5
[6E2] 2185: ;Für Output
[DS2] 2186: X1F0X: MOV CL,20H
[GT1] 2187: MOV AH,3CH
[N81] 2188: INT 21H
[B93] 2189: ;Betriebsart nach AH
[CN3] 2190: X1F1: MOV DS:[BX-2]
[L32] 2191: ;Dateinummer
[H13] 2192: MOV BX,DS:[BX-4]
[G71] 2193: POP DS
[DP2] 2194: ;Handle sichern
[NL2] 2195: MOV DS:[BX],AL
[XV3] 2196: ;Positionszeiger auf 0
[Y24] 2197: MOV BYTE PTR DS:[4*BX],0
[GJ3] 2198: MOV BYTE PTR DS:[16*BX],26
[394] 2199: ;Letztes Zeichen = EOF
[P95] 2200: ;Pseudo-Handles, dann Schluß
[8L1] 2201: MOV AL,-3
[AF1] 2202: JAE X1F6
[S22] 2203: ;Für Append?
[AD1] 2204: CMP AH,3
[SG1] 2205: JNE X1F2
[X44] 2206: ;Zeiger auf das Dateiende
[YT1] 2207: MOV
[AW1] 2208: XOR CX,CX
[HW1] 2209: XOR DX,DX
[312] 2210: MOV AX,4202H
[N81] 2211: INT 21H
```



```
[2N2] 2212: JMP SHORT X1F4
[7A2] 2213: ;Für Input?
[X72] 2214: X1F2: CMP AH,0
[8A2] 2215: JNE X1F4
[574] 2216: MOV BYTE PTR DS:[16+BX],0
[SR3] 2217: ;Erstes Zeichen lesen
[D61] 2218: MOV DX,X00W-L00
[DE1] 2219: CALL DX
[2N2] 2220: JMP SHORT X1F4
[KB2] 2221: ;Für Random?
[2B2] 2222: X1F3: CMP AH,2
[9A1] 2223: JNE X1F4
[L91] 2224: PUSH BX
[PP2] 2225: MOV BL,DS:[BX]
[XA4] 2226: ;Zeiger auf das Dateideinde
[XM1] 2227: XOR CX,CX
[XM1] 2228: XOR DX,DX
[312] 2229: MOV AX,4202H
[NB1] 2230: INT 21H
[D71] 2231: POP BX
[PA1] 2232: JC X1F4
[2G3] 2233: ;Datei 0 Byte groß?
[MN1] 2234: OR AX,DX
[WAL] 2235: JE X1F4
[574] 2236: MOV BYTE PTR DS:[16+BX],0
[S12] 2237: X1F4: MOV BP,SP
[3B3] 2238: ;Satzlänge sichern
[813] 2239: MOV AL,SS:[BP+2]
[953] 2240: MOV DS:[26+BX],AL
[X12] 2241: X1F6: RET 62
[DE3] 2242: ;Fehler beim Öffnen
[NR3] 2243: X1F5: MOV BX,DS:[BX-4]
[G71] 2244: POP DS
[K03] 2245: MOV BYTE PTR DS:[BX],0
[Y24] 2246: MOV BYTE PTR DS:[4+BX],0
[394] 2247: MOV BYTE PTR DS:[16+BX],26
[571] 2248: RET 262
[000] 2249: ;CLOSE
[XE2] 2251: L20: MOV BP,SP
[743] 2252: ;Dateinummer holen
[S13] 2253: MOV BX,SS:[BP+2]
[DVS] 2254: ;Unterprogramm schließe Datei BX
[MK2] 2255: MOV DX,X00W-L00
[D61] 2256: CALL DX
[B21] 2257: RET 2
[000] 2258:
[YE4] 2259: ;PRINT 14er-Tabulator (" ")
[1F2] 2260: L21: MOV BP,SP
[S13] 2261: MOV BX,SS:[BP+2]
[W32] 2262: ;Bildschirm?
[2V3] 2263: CMP BYTE PTR DS:[BX],-1
[PE1] 2264: JNE X210
[EB2] 2265: ;Cursor holen
[7R1] 2266: MOV AH,03H
[GB1] 2267: INT 19H
[3G2] 2268: MOV CX,DS:[11]
[RL1] 2269: SUB AL,23
[1P4] 2270: ;Bleibt Cursor in der Zeile?
[MS1] 2271: CMP DL,AL
[191] 2272: JB X21
[AA3] 2273: ;In neue Zeile gehen
[6N1] 2274: MOV AL,13
[AG2] 2275: MOV DX,X006-L00
[371] 2276: JMP DX
[AK3] 2277: X210: MOV DL,DS:[4+BX]
[C35] 2278: ;Nächste Position berechnen
[7F2] 2279: X211: MOV AL,DL
[LE1] 2280: MOV AH,0
[YN1] 2281: MOV CX,14
[L61] 2282: DIV CL
[0S1] 2283: SUB CL,AH
[D16] 2284: ;Bereich mit Leerzeichen auffüllen
[8D2] 2285: X212: MOV AL,0
[R91] 2286: PUSH CX
[AG2] 2287: MOV DX,X006-L00
[D61] 2288: CALL DX
[H71] 2289: POP CX
[VH1] 2290: LOOP X212
[3P0] 2291: RET
[000] 2292:
[0P2] 2293: ;LINE INPUT
[H12] 2294: L22: POP AX
[8W1] 2295: MOV BP,SP
[W44] 2296: ;Adresse des Strings holen
[PR2] 2297: MOV ES,SS:[BP]
[N25] 2298: ;Rücksperrungsadresse auf Stack
[NM2] 2299: MOV SS:[BP],AX
[3F1] 2300: MOV DI,1
[L32] 2301: ;Dateinummer
[S13] 2302: MOV BX,SS:[BP+2]
[TR3] 2303: MOV BYTE PTR DS:[14],0
[N72] 2304: X220: PUSH DI
[8G4] 2305: ;Zeichen aus Datei BX lesen
[CK2] 2306: MOV DX,X00W-L00
[D61] 2307: CALL DX
[671] 2308: POP DI
[L53] 2309: ;Nullen überlesen
[5D1] 2310: CMP AL,0
[991] 2311: JE X220
[202] 2312: ;End of File
[3M1] 2313: CMP AL,26
[F91] 2314: JE X222
[ML1] 2315: CMP AL,13
[C91] 2316: JE X221
[WA2] 2317: ;String voll?
[YN1] 2318: CMP DI,256
[HD1] 2319: JAB X220
[XN0] 2320: STOSB
[1G1] 2321: JMP X220
[B14] 2322: ;Bildschirm, dann fertig
[BK4] 2323: X221: CMP BYTE PTR DS:[BX],-1
[F91] 2324: JE X222
[U07] 2325: ;Datei, dann eventuell nachfolgendes LF les
en
[774] 2326: CMP BYTE PTR DS:[16+BX],10
[YE1] 2327: JNE X222
[C91] 2328: PUSH DI
[CK2] 2329: MOV DX,X00W-L00
[D61] 2330: CALL DX
[671] 2331: POP DI
[F04] 2332: ;Länge des Strings sichern
[853] 2333: X222: LEA AX,[DI-1]
[L22] 2334: MOV ES:[0],AL
[AW6] 2335: ;Eventuell in neue Zeile gehen (SCRN*)
[XK2] 2336: MOV DX,X00S-L00
[371] 2337: JMP DX
[000] 2338:
[X15] 2339: ;PRINT " " (bei INPUT*)
[YF2] 2340: L23: SUB SP,254
[7G4] 2341: ;Dateinummer 0 = Bildschirm
[BP1] 2342: MOV AL," "
[AG2] 2343: MOV DX,X006-L00
[D61] 2344: CALL DX
[FK1] 2345: MOV AL," "
[AG2] 2346: MOV DX,X006-L00
[371] 2347: JMP DX
[000] 2348:
[5B3] 2349: ;INPUT String
[LH2] 2350: L24: SUB SP,254
[2G4] 2351: ;Platz für String schaffen
[8W1] 2352: MOV BP,SP
[XA4] 2353: ;Rücksperrungsadresse sichern
[XR2] 2354: PUSH SS:[BP+254]
[SW1] 2355: MOV AX,SS
[1W1] 2356: MOV ES,AX
[2D2] 2357: LEA DI,[BP+1]
[L32] 2358: ;Dateinummer
[X73] 2359: MOV BX,SS:[BP+258]
[G21] 2360: ;Input-Flag
```

```
[104] 2361: AND BYTE PTR DS:[14],7FH
[232] 2362: X240: CALL X248
[XA1] 2363: JE X247
[V36] 2364: ;Führende Leerzeichen überlesen
[XN1] 2365: CMP AL," "
[D91] 2366: JE X240
[B95] 2367: ;Beginnt String mit Hochkomma?
[9B2] 2368: X241: CMP AL,34
[771] 2369: JNE X244
[978] 2370: ;Ja, dann lesen bis zweites Hochkomma oder
Zeilenende
[9A2] 2371: CALL X248
[2A1] 2372: JE X247
[Y11] 2373: CMP AL,34
[P91] 2374: JE X243
[C91] 2375: PUSH DI
[471] 2376: SUB DI,BP
[WA2] 2377: ;String voll?
[YN1] 2378: CMP DI,256
[671] 2379: POP DI
[V01] 2380: JAB X242
[NX0] 2381: STOSB
[BG1] 2382: JMP X242
[7W6] 2383: ;Nächstes Komma oder Zeilenende suchen
[AA2] 2384: X243: CALL X248
[2A1] 2385: JE X247
[DK1] 2386: CMP AL,","
[5F1] 2387: JNE X243
[912] 2388: JMP SHORT X247
[6M6] 2389: ;Zeichen lesen bis Komma oder Zeilenende
[U02] 2390: X244: CMP AL,","
[Z91] 2391: JE X246
[C91] 2392: PUSH DI
[471] 2393: SUB DI,BP
[WA2] 2394: ;String voll?
[YN1] 2395: CMP DI,256
[671] 2396: POP DI
[4E1] 2397: JAB X245
[NX0] 2398: STOSB
[CA2] 2399: X245: CALL X248
[8F1] 2400: JNE X244
[RA6] 2401: ;Nachfolgende Leerzeichen entfernen
[702] 2402: X246: DEC DI
[471] 2403: CMP DI,BP
[3B1] 2404: JE X248
[CV3] 2405: CMP BYTE PTR DS:[DI],32
[Z91] 2406: JE X246
[A12] 2407: X248: INC DI
[4A4] 2408: ;Länge des Strings sichern
[N53] 2409: X247: LEA AX,[DI-1]
[771] 2410: SUB AX,BP
[9N2] 2411: MOV SS:[BP],AL
[3P0] 2412: RET
[AW5] 2413: ;Unterprogramm "ein Zeichen lesen"
[S62] 2414: X248: PUSH DI
[AK2] 2415: MOV DX,X00W-L00
[D61] 2416: CALL DX
[671] 2417: POP DI
[3M1] 2418: CMP AL,26
[9B1] 2419: JE X248
[8BA] 2420: ;Sonderbehandlung "Carriage Return" und "Li
ne Feed"
[ML1] 2421: CMP AL,13
[XF1] 2422: JNE X24A
[7A4] 2423: CMP BYTE PTR DS:[16+BX],10
[RF1] 2424: JNE X249
[C91] 2425: PUSH DI
[2B4] 2426: ;Line Feed überlesen
[CK2] 2427: MOV DX,X00W-L00
[D61] 2428: CALL DX
[671] 2429: POP DI
[YN1] 2430: MOV AL,13
[AB2] 2431: X249: CMP AL,13
[3C1] 2432: X24A: RET
[000] 2433:
[YN1] 2434: ;LEA
[HP2] 2435: L25: MOV BP,SP
[K66] 2436: ;Länge des Strings ins Integerformat
[813] 2437: MOV AL,SS:[BP+2]
[LE1] 2438: MOV AH,0
[B53] 2439: MOV SS:[BP+256],AX
[871] 2440: RET 254
[000] 2441:
[065] 2442: ;CLOSE: alle offenen Dateien
[772] 2443: MOV BX,4
[D53] 2444: ;Schließe Datei BX
[HA3] 2445: X260: MOV DX,X00X-L00
[D61] 2446: CALL DX
[341] 2447: DEC BX
[0P1] 2448: JNE X260
[3P0] 2449: RET
[000] 2450:
[L25] 2451: ;Funktion "EOF" (End of File)
[TP2] 2452: L27: MOV BP,SP
[L32] 2453: ;Dateinummer
[S13] 2454: MOV BX,SS:[BP+2]
[W32] 2455: ;Bildschirm?
[XM1] 2456: MOV AL,-1
[3K2] 2457: CMP DS:[BX],AL
[L91] 2458: JE X270
[V27] 2459: ;Nächstes zu lesendes Zeichen = "EOF"?
[H74] 2460: CMP BYTE PTR DS:[16+BX],26
[P91] 2461: JE X271
[L32] 2462: X270: NOT AL
[LB1] 2463: X271: CBW
[L75] 2464: ;Ergebnis der Funktion auf Stack
[H13] 2465: MOV SS:[BP+2],AX
[3P0] 2466: RET
[000] 2467:
[CT3] 2468: ;Funktion "INKEYS"
[3K2] 2469: L28: SUB SP,254
[2G4] 2470: ;Platz für String schaffen
[8W1] 2471: MOV BP,SP
[GN3] 2472: ;Hole Tastaturstatus
[3T1] 2473: MOV AH,0BH
[NB1] 2474: INT 21H
[EM1] 2475: MOV AH,-1
[N72] 2476: ;Taste gedrückt?
[5D1] 2477: CMP AL,0
[N91] 2478: JE X280
[H21] 2479: ;Taste lesen
[VH1] 2480: MOV AH,0BH
[NB1] 2481: INT 21H
[PT1] 2482: MOV AH,AL
[WA2] 2483: ;Stringlänge
[U01] 2484: MOV AL,1
[AM3] 2485: X280: MOV SS:[BP],AX
[694] 2486: ;Erweiterter Tastencode?
[LD1] 2487: CMP AH,0
[7F1] 2488: JNE X281
[DN2] 2489: ;Scancode lesen
[VH1] 2490: MOV AH,0BH
[NB1] 2491: INT 21H
[Q93] 2492: MOV SS:[BP+2],AL
[XA5] 2493: ;String jetzt zwei Zeichen lang
[UG3] 2494: INC BYTE PTR SS:[BP]
[NF3] 2495: X281: JMP SS:[BP+254]
[000] 2496:
[DE6] 2497: ;Zwei Strings miteinander vergleichen
[S63] 2498: SVGL MACRO MASK,CMD
[YF1] 2499: LOCAL LBL
[C98] 2500: ;Unterprogramm "Strings vergleichen" aufruf
en
[BK2] 2501: MOV DX,X00U-L00
[D61] 2502: CALL DX
[U74] 2503: ;Ergebnis initialisieren
[HW1] 2504: XOR DX,DX
[304] 2505: ;Vergleich durchführen
[X92] 2506: TEST AL,MASK AND 7
[7A1] 2507: CWD LBL
[B41] 2508: DEC DX
```

```
[7B3] 2509: ;Ergebnis auf Stack
[754] 2510: LBL: MOV SS:[BP+510],DX
[404] 2511: ;Beide Strings entfernen
[871] 2512: RET 510
[HP0] 2513: ENDM
[000] 2514:
[GP2] 2515: ;*STRING ==
[772] 2516: L29: SVGL 1,JE
[000] 2517:
[DP2] 2518: ;*STRING <
[AB2] 2519: L2A: SVGL 2,JE
[000] 2520:
[LP2] 2521: ;*STRING >
[M82] 2522: L2B: SVGL 4,JE
[000] 2523:
[RD2] 2524: ;*STRING <=
[N82] 2525: L2C: SVGL 3,JE
[000] 2526:
[XD2] 2527: ;*STRING >=
[Z82] 2528: L2D: SVGL 5,JE
[000] 2529:
[V02] 2530: ;*STRING <>
[SP2] 2531: L2E: SVGL 1,JNE
[000] 2532:
[Z32] 2533: ;*INPUTS*
[V32] 2534: L2F: POP SI
[D71] 2535: POP BX
[N71] 2536: POP CX
[2G4] 2537: ;Platz für String schaffen
[1S1] 2538: SUB SP,256
[8W1] 2539: MOV BP,SP
[HN2] 2540: MOV SS:[BP],CL
[EUI] 2541: ;Anzahl = 0?
[8L1] 2542: JCXZ X2F1
[861] 2543: ;Lesen erlauben, wenn vorher "EOF"
[SU3] 2544: MOV BYTE PTR DS:[14],-1
[P32] 2545: X2F0: INC BP
[R91] 2546: PUSH CX
[8G4] 2547: ;Zeichen aus Datei BX lesen
[CK2] 2548: MOV DX,X00W-L00
[D61] 2549: CALL DX
[N71] 2550: POP CX
[Q42] 2551: ;End of File?
[3M1] 2552: CMP AL,26
[MA1] 2553: JE X2F1
[PT2] 2554: ;Zeichen sichern
[N25] 2555: MOV SS:[BP],AL
[MK2] 2556: ;CX Zeilen lesen
[1L2] 2557: LOOP X2F0
[SL2] 2558: X2F1: MOV BP,SP
[R87] 2559: ;Wirklich gelesene Zeichen als Stringlänge
[L22] 2560: SUB SS:[BP],CL
[G81] 2561: JMP SI
[000] 2562:
[T95] 2563: ;Nach Eingabe in neue Zeile gehen
[0F2] 2564: L30: MOV BP,SP
[L32] 2565: ;Dateinummer
[S13] 2566: MOV BX,SS:[BP+2]
[7K2] 2567: MOV DX,X00S-L00
[371] 2568: JMP DX
[000] 2569:
[3M3] 2570: ;Funktion "INST"
[YF2] 2571: L31: MOV BP,SP
[MY3] 2572: ;Länge des Suchstrings
[N73] 2573: MOV CL,SS:[BP+258]
[F25] 2574: ;Länge des Suchstrings
[C13] 2575: MOV CH,SS:[BP+2]
[9D1] 2576: CMP CH,0
[H91] 2577: JE X313
[MY3] 2578: ;Anfangsposition holen
[773] 2579: MOV BX,SS:[BP+514]
[5F2] 2580: X310: MOV BP,BL
[L31] 2581: DEC AL
[TU6] 2582: ;Suchstring zu lang, dann nicht gefunden
[FN1] 2583: ADD AL,CH
[3C1] 2584: JC X313
[ES1] 2585: CMP AL,CL
[591] 2586: JA X313
[F25] 2587: ;Zeichenzeiger initialisieren
[CK1] 2588: MOV SI,BX
[3F1] 2589: MOV DI,1
[WT1] 2590: MOV DL,CH
[X12] 2591: X311: INC SI
[151] 2592: INC DI
[6B4] 2593: ;Zeichenweise vergleichen
[CS3] 2594: MOV AL,SS:[BP+257+SI]
[HB3] 2595: CMP AL,SS:[BP+1+DI]
[XE1] 2596: JNE X312
[Z31] 2597: DEC DL
[U01] 2598: JNE X311
[MK2] 2599: JMP SHORT X314
[CG6] 2600: ;Position verschieben, neu vergleichen
[F22] 2601: L32: INC BX
[0G1] 2602: JMP X310
[YH2] 2603: X313: XOR BX,BX
[7B3] 2604: ;Ergebnis auf Stack
[G34] 2605: X314: MOV SS:[BP+514],BX
[CT1] 2606: RET 512
[000] 2607:
[295] 2608: ;Stringzuweisung in "DEF FN"
[F22] 2609: L33: MOV SI,SP
[RV1] 2610: MOV DX,DS
[SW1] 2611: MOV AX,SS
[XV1] 2612: MOV DS,AX
[PW0] 2613: LODSW
[PW0] 2614: LODSW
[1W1] 2615: MOV ES,AX
[G01] 2616: XOR DI,DI
[XA4] 2617: ;String in Speicher kopieren
[PW0] 2618: TRANS
[6W1] 2619: MOV DS,DX
[G71] 2620: RET 258
[000] 2621:
[2P4] 2622: ;Rücksperrung nach "FN"
[CF2] 2623: L33: MOV BP,SP
[7B3] 2624: ;Ergebnis auf Stack
[M13] 2625: MOV AX,SS:[BP+2]
[Q93] 2626: XCHG AX,SS:[BP+4]
[YL5] 2627: ;Rücksperrungsadressen entfernen
[ED1] 2628: ADD SP,4
[R61] 2629: JMP AX
[000] 2630:
[0P4] 2631: ;Rücksperrung nach "FN"
[7G2] 2632: L34: MOV SI,SP
[1D1] 2633: ADD SI,2
[RV3] 2634: MOV DX,DS
[SW1] 2635: MOV AX,SS
[XV1] 2636: MOV DS,AX
[1W1] 2637: MOV ES,AX
[FH4] 2638: ;Rücksperrungsadresse holen
[263] 2639: MOV BX,DS:[SI+256]
[326] 2640: ;String um 2 Bytes nach oben schieben
[HP2] 2641: MOV CL,DS:[SI]
[VE1] 2642: MOV CH,0
[MT1] 2643: ADD SI,CX
[CF2] 2644: LEA DI,[SI+2]
[Z45] 2645: INC CX
[KN0] 2646: STD
[XK1] 2647: REP MOVSB
[C17] 2648: ;Standard-Transferrichtung wieder setzen
[UL0] 2649: CLD
[6W1] 2650: MOV DS,DX
[ML4] 2651: ;Rücksperrungsadressen weg
[ED1] 2652: ADD SP,4
```

Listing 2. Der Quelltext der SBasic-Laufzeitbibliothek »sbasic.asm«


```
[V61] 2653: JMP BX
[000] 2654:
[12] 2655: ;POKE
[MF2] 2656: L35: MOV BP,SP
[VR3] 2657: ;Segmentadresse holen
[DH2] 2658: MOV ES,DS:[25]
[FD1] 2659: ;Adresse
[N13] 2660: MOV DI,SS:[BP+4]
[9W0] 2661: ;Wert
[813] 2662: MOV AL,SS:[BP+2]
[NX0] 2663: STOSB
[121] 2664: RET 4
[000] 2665:
[9A3] 2666: ;Funktion =PEEK
[SF2] 2667: L36: MOV BP,SP
[VR3] 2668: ;Segmentadresse holen
[DH2] 2669: MOV ES,DS:[25]
[FD1] 2670: ;Adresse
[H13] 2671: MOV DI,SS:[BP+2]
[CM2] 2672: MOV AL,ES:[DI]
[LE1] 2673: MOV AH,0
[N53] 2674: ;Wert zurückgeben
[H13] 2675: MOV SS:[BP+2],AX
[3P0] 2676: RET
[000] 2677:
[FG3] 2678: ;Funktion =ASC
[WF2] 2679: L37: MOV BP,SP
[W13] 2680: MOV CX,SS:[BP+2]
[WV1] 2681: XOR AX,AX
[E73] 2682: ;Länge des Strings
[GD1] 2683: CMP CL,1
[891] 2684: JB X370
[GT1] 2685: MOV AH,CH
[F44] 2686: X370: MOV SS:[BP+256],AX
[871] 2687: RET 254
[000] 2688:
[GG3] 2689: ;Funktion =LETS
[GC2] 2690: L38: MOV BP,SP
[WAZ] 2691: ;Stringlänge
[C13] 2692: MOV AL,SS:[BP+4]
[PZ1] 2693: ;Neue Länge
[413] 2694: MOV AH,SS:[BP+2]
[AS1] 2695: CMP AH,AL
[YD1] 2696: JAE X390
[AT1] 2697: MOV AL,AH
[UL4] 2698: ;Länge in String schreiben
[8R3] 2699: X390: MOV SS:[BP+4],AL
[B21] 2700: RET 2
[000] 2701:
[ET3] 2702: ;Funktion =RIGHTS
[EP2] 2703: L39: MOV DI,SP
[B81] 2704: ADD DI,4
[PZ1] 2705: ;Neue Länge
[W03] 2706: MOV CL,SS:[DI-2]
[WAZ] 2707: ;Stringlänge
[DP2] 2708: MOV BL,SS:[DI]
[BK6] 2709: ;Anzahl der zu überspringenden Zeichen
[KS1] 2710: SUB BL,CL
[SE1] 2711: JBE X390
[RE1] 2712: MOV BH,0
[X22] 2713: LEA SI,[DI+BX+1]
[RV1] 2714: MOV DX,DS
[SW1] 2715: MOV AX,SS
[XV1] 2716: MOV DS,AX
[1W1] 2717: MOV ES,AX
[VE1] 2718: MOV CH,0
[W43] 2719: ;Neue Stringlänge
[G01] 2720: MOV AL,CL
[NX0] 2721: STOSB
[RX2] 2722: ;String kopieren
[X11] 2723: REP MOVSB
[6W1] 2724: MOV DS,DX
[XR1] 2725: X390: RET 2
[000] 2726:
[G43] 2727: ;Funktion =MIDS
[EG2] 2728: L3A: MOV DI,SP
[H81] 2729: ADD DI,6
[RV1] 2730: MOV DX,DS
[SW1] 2731: MOV AX,SS
[XV1] 2732: MOV DS,AX
[1W1] 2733: MOV ES,AX
[DR2] 2734: ;Erste Position
[B22] 2735: MOV BL,DS:[DI-2]
[9D1] 2736: CMP BL,0
[RF1] 2737: JNE X3A0
[VE1] 2738: MOV BL,1
[9G4] 2739: ;Zeiger auf erstes Zeichen
[G82] 2740: X3A0: MOV BH,0
[VR2] 2741: LEA SI,[DI+BX]
[G43] 2742: ;Alte Stringlänge
[4N2] 2743: MOV CL,DS:[DI]
[VE1] 2744: MOV CH,0
[S55] 2745: ;Position des Strings außerhalb?
[R31] 2746: DEC BX
[LS1] 2747: SUB CL,BL
[LE1] 2748: JAE X3A1
[ZE1] 2749: MOV CL,0
[ND3] 2750: ;Anzahl der Zeichen
[H3] 2751: X3A1: MOV AL,DS:[DI-4]
[G81] 2752: CMP AL,1
[TE1] 2753: JBE X3A2
[1F8] 2754: ;Es sind weniger Zeichen vorhanden als gewu
nscht
[ZU1] 2755: MOV CL,AL
[E73] 2756: ;Länge des Strings
[9H2] 2757: X3A2: MOV AL,CL
[NX0] 2758: STOSB
[ST1] 2759: ;Zeichen kopieren
[X11] 2760: REP MOVSB
[6W1] 2761: MOV DS,DX
[LE1] 2762: RET 4
[000] 2763:
[W54] 2764: ;Funktion =STRINGS
[KG2] 2765: L3B: MOV DI,SP
[B81] 2766: ADD DI,4
[SW1] 2767: MOV AX,SS
[1W1] 2768: MOV ES,AX
[EM4] 2769: ;Anzahl der Wiederholungen
[S53] 2770: MOV CL,ES:[DI+254]
[VE1] 2771: MOV CH,0
[PZ2] 2772: MOV AX,ES:[DI-2]
[U94] 2773: ;Länge des Quellstrings
[5D1] 2774: CMP AL,0
[TF1] 2775: JNE X3B0
[ZE1] 2776: MOV CL,0
[SN3] 2777: ;Stringlänge strecken
[BH2] 2778: X3B0: MOV AL,CL
[NX0] 2779: STOSB
[V33] 2780: ;Zeichen schreiben
[AT1] 2781: MOV AL,AH
[KX1] 2782: REP STOSB
[B21] 2783: RET 2
[000] 2784:
[6W9] 2785: ;VARPTR= String (Konvertierung in das GW-B
ASIC-Format)
[CH2] 2786: L3C: MOV BP,SP
[CL2] 2787: ;Stringsegment
[S13] 2788: MOV AX,SS:[BP+4]
[1W1] 2789: MOV ES,AX
[DS6] 2790: ;Offset des Strings zu DS berechnen
[FV1] 2791: MOV BX,DS
[7U1] 2792: SUB AX,BX
[GC3] 2793: ;Offset x 4 = Segment
[BP1] 2794: MOV CL,4
[V81] 2795: SHL AX,CL
[D44] 2796: ;Längenbyte überlesen
[451] 2797: INC AX
[P32] 2798: ;Descriptor
[S13] 2799: MOV BX,SS:[BP+2]
[943] 2800: ;Adresse schreiben
```

```
[UZ2] 2801: MOV DS:[BX+1],AX
[D93] 2802: ;Längenbyte lesen
[PK2] 2803: MOV AL,ES:[0]
[NL2] 2804: MOV DS:[BX],AL
[EP3] 2805: ;Descriptor auf Stack
[T13] 2806: MOV SI:[BP+4],BX
[B21] 2807: RET 2
[000] 2808:
[492] 2809: ;CALL
[GH2] 2810: L3D: MOV BP,SP
[RS6] 2811: ;Segment für *Far Return* auf Stack
[BV1] 2812: MOV AX,CS
[N03] 2813: XCHG AX,SS:[BP+2]
[T25] 2814: ;Far-Sprung ins BASIC-Defensegment
[FV1] 2815: MOV BX,DS
[4W1] 2816: MOV ES,BX
[L91] 2817: PUSH BX
[F91] 2818: PUSH AX
[D71] 2819: DB 0CH
[000] 2820:
[KV3] 2821: ;MIDS* (als Befehl)
[BK2] 2822: L3E: MOV SI,SP
[1D1] 2823: ADD SI,2
[RV1] 2824: MOV DX,DS
[SW1] 2825: MOV AX,SS
[XV1] 2826: MOV DS,AX
[N03] 2827: ;Anzahl der Zeichen
[R63] 2828: MOV CL,DS:[SI+256]
[VE1] 2829: MOV CH,0
[U94] 2830: ;Länge des Quellstrings
[C90] 2831: LODSB
[ES1] 2832: CMP AL,CL
[SE1] 2833: JAE X3E0
[ZU1] 2834: MOV CL,AL
[F84] 2835: ;Segment des Quellstrings
[154] 2836: X3E0: MOV ES,DS:[SI+259]
[HP2] 2837: ;Startposition
[R63] 2838: MOV BL,DS:[SI+257]
[RE1] 2839: MOV BH,0
[9D1] 2840: CMP BL,0
[7A1] 2841: JC X3E1
[B51] 2842: INC BX
[RK2] 2843: X3E1: MOV DI,BX
[Y33] 2844: ;Länge des Zielstrings
[PK2] 2845: MOV AL,ES:[0]
[TS3] 2846: ;Position außerhalb?
[BS1] 2847: SUB AL,BL
[LAI] 2848: JC X3E3
[S41] 2849: INC AL
[W66] 2850: ;Zielstringlänge muß gleich bleiben
[G51] 2851: CMP CL,AL
[IF1] 2852: JBE X3E2
[SY2] 2853: ;Zeichen kopieren
[ZU1] 2854: MOV CL,AL
[ML2] 2855: X3E2: REP MOVSB
[SW2] 2856: X3E3: MOV DS,DX
[S71] 2857: RET 262
[000] 2858:
[GG3] 2859: ;Funktion =TIME$
[VL2] 2860: L3F: SUB SP,254
[ZG4] 2861: ;Platz für String schaffen
[8W1] 2862: MOV BP,SP
[KAA] 2863: ;Zeit übers MS-DOS lesen
[DT1] 2864: MOV AH,2CH
[N81] 2865: INT 21H
[SG3] 2866: ;Länge von *TIME$*
[TU3] 2867: MOV BYTE PTR SS:[BP],0
[YE1] 2868: ;Stunden
[GT1] 2869: MOV AL,CH
[MGI] 2870: CALL X3F0
[R13] 2871: MOV SS:[BP+1],AX
[EC2] 2872: ;Trennzeichen
[C64] 2873: MOV BYTE PTR SS:[BP+3],""
[RE1] 2874: ;Minuten
[Q11] 2875: MOV AL,CL
[MGI] 2876: CALL X3F0
[R13] 2877: MOV SS:[BP+4],AX
[S64] 2878: MOV BYTE PTR SS:[BP+6],""
[LK1] 2879: ;Sekunden
[LT1] 2880: MOV AL,DH
[MGI] 2881: CALL X3F0
[Q23] 2882: MOV SS:[BP+7],AX
[SW2] 2883: CMP SI:[BP+254]
[R68] 2884: ;Unterprogramm *Rechnen AL in ASCII-Darstell
ung um*
[192] 2885: X3F0: MOV AH,0
[F61] 2887: DIV BL,10
[CL1] 2888: ADD AX,00H
[3P0] 2889: RET
[000] 2890:
[CS1] 2891: ;FIELD
[3F2] 2892: L40: MOV BP,SP
[343] 2893: ;Anzahl der Felder
[R13] 2894: MOV AL,SS:[BP+2]
[DR3] 2895: ;Adresse der Felddatei
[H33] 2896: MOV SI,SS:[BP+4]
[L32] 2897: ;Datennummer
[Q23] 2898: MOV BX,SS:[BP+6]
[ZS2] 2899: ;Anzahl sichern
[AN2] 2900: MOV DS:[SI],AL
[TD1] 2901: SHL BX,1
[4U2] 2902: ;Adresse sichern
[N63] 2903: MOV DS:[SI],AL
[343] 2904: ;Anzahl der Felder
[C90] 2905: LODSB
[G01] 2906: MOV DL,AL
[UF3] 2907: ;Länge eines Feldes
[9R1] 2908: X400: MOV SI,DS:[BP+4]
[LE1] 2909: MOV AH,0
[YV1] 2910: MOV CX,AX
[DT3] 2911: ;Segment eines Strings
[PW0] 2912: LODSW
[W36] 2913: ;String mit Leerzeichen vorbelegen
[1W1] 2914: MOV ES,AX
[TF2] 2915: MOV ES:[0],CL
[3F1] 2916: MOV DI,1
[FK1] 2917: MOV AL," "
[KX1] 2918: REP STOSB
[484] 2919: ;Nächste Felddefinition
[Z31] 2920: DEC DL
[PE1] 2921: JNE X400
[V21] 2922: RET 6
[000] 2923:
[PM1] 2924: ;GET*
[F22] 2925: L41: MOV BP,SP
[L32] 2926: ;Datennummer
[Q23] 2927: MOV BX,SS:[BP+6]
[Z34] 2928: ;Adresse Feldbeschreiber
[TD1] 2929: SHL BX,1
[N93] 2930: MOV SI,DS:[28+BX]
[HE1] 2931: SHR BX,1
[343] 2932: ;Anzahl der Felder
[R73] 2933: MOV AL,DS:[26+BX]
[LE1] 2934: MOV AH,0
[F91] 2935: PUSH AX
[193] 2936: ;Datensatz-Nummer
[Q23] 2937: MOV DX,SS:[BP+2]
[Q23] 2938: MOV CX,SS:[BP+4]
[L27] 2939: ;Vom Single- ins Long-Integer-Format wandel
n
[MH1] 2940: OR CL,00H
[ZH2] 2941: X413: CMP CH,152
[XD1] 2942: JAE X414
[9E1] 2943: SHR CL,1
[B81] 2944: RCR DX,1
[W41] 2945: INC CH
[AG1] 2946: JNB X413
[N72] 2947: X414: SUB DX,1
[MB1] 2948: SBB CL,0
```

```
[171] 2949: MUL CL
[VV1] 2950: MOV CX,AX
[UT6] 2951: ;Mit Länge eines Feldes multiplizieren
[971] 2952: POP AX
[F91] 2953: PUSH AX
[H71] 2954: MUL DX
[7S1] 2955: ADD DX,CX
[ZW1] 2956: MOV DX,AX
[PV1] 2957: MOV DI,BX
[D22] 2958: ;Datei-Handle
[PP2] 2959: MOV BL,DS:[BX]
[S25] 2960: ;Schreib-/Leserzeiger positionieren
[902] 2961: MOV AX,4200H
[N81] 2962: INT 21H
[H71] 2963: POP CX
[C91] 2964: PUSH DI
[C35] 2965: ADD AX,CX
[V81] 2966: ;Nächste Position berechnen
[AB1] 2967: ADC DX,0
[VW1] 2968: MOV ES,AX
[VW1] 2969: MOV BP,DX
[343] 2970: ;Anzahl der Felder
[C90] 2971: LODSB
[LE1] 2972: MOV AH,0
[LVI] 2973: MOV DI,AX
[BW3] 2974: ;Länge eines Strings
[DR1] 2975: X410: LODSW
[VV1] 2976: MOV CX,AX
[P91] 2977: PUSH DS
[XS3] 2978: ;Adresse eines Strings
[PW0] 2979: LODSW
[XV1] 2980: MOV DS,AX
[WAZ] 2981: ;Stringlänge
[NF2] 2982: MOV DS:[0],CL
[UK2] 2983: ;Aus Datei lesen
[RF1] 2984: MOV DX,CL
[WT1] 2985: MOV AH,3FH
[N81] 2986: INT 21H
[G71] 2987: POP DS
[KC1] 2988: ;Fehler?
[991] 2989: JC X412
[AU1] 2990: CMP AX,CX
[YE1] 2991: JNE X412
[HL3] 2992: ;Nächstes Feld lesen
[W31] 2993: DEC DI
[SE1] 2994: JNE X410
[SP3] 2995: ;Zeiger ans Dateiteile
[AW1] 2996: XOR CX,CX
[RW1] 2997: XOR DX,DX
[312] 2998: MOV AX,4202H
[N81] 2999: INT 21H
[991] 3000: JC X412
[6W7] 3001: ;Zeiger mit gesicherter Position vergleiche
n
[G01] 3002: CMP BP,DX
[V81] 3003: JNE X411
[XV1] 3004: MOV SI,ES
[XV1] 3005: CMP SI,AX
[F91] 3006: JE X412
[T32] 3007: X411: POP BX
[XV1] 3008: MOV BYTE PTR DS:[16+BX],0
[V21] 3009: RET
[U32] 3010: X412: POP BX
[G04] 3011: ;Dateiteile (=EOF) erreicht
[394] 3012: MOV BYTE PTR DS:[16+BX],26
[V21] 3013: RET 6
[000] 3014:
[0R5] 3015: ;Stringzuweisung mit *LSET*
[ZG2] 3016: L42: MOV SI,SP
[1D1] 3017: ADD SI,2
[RV1] 3018: MOV DX,DS
[SW1] 3019: MOV AX,SS
[XV1] 3020: MOV DS,AX
[F84] 3021: ;Segment des Zielstrings
[G73] 3022: MOV DS:[SI+256]
[3F1] 3023: MOV DI,1
[WAZ] 3024: ;Stringlänge
[TK2] 3025: MOV CL,ES:[0]
[VE1] 3026: MOV CH,0
[VE1] 3027: MOV BL,0
[U94] 3028: ;Länge des Quellstrings
[C90] 3029: LODSB
[AC3] 3030: ;Länger, dann abschneiden
[ES1] 3031: CMP AL,CL
[LDI] 3032: JAE X420
[Z33] 3033: ;Längendifferenz
[AU1] 3034: MOV BL,CL
[CS1] 3035: SUB BL,AL
[ZU1] 3036: MOV CL,AL
[UN3] 3037: ;Zeichen übertragen
[9H2] 3038: X420: REP MOVSB
[EUA] 3039: ;Rest mit Leerzeichen füllen
[SU1] 3040: MOV AL," "
[FK1] 3041: MOV AL," "
[KX1] 3042: REP STOSB
[6W1] 3043: MOV DS,DX
[G71] 3044: RET 258
[000] 3045:
[F85] 3046: ;Stringzuweisung per *RSET*
[G02] 3047: L43: MOV SI,SP
[1D1] 3048: ADD SI,2
[RV1] 3049: MOV DX,DS
[SW1] 3050: MOV AX,SS
[XV1] 3051: MOV DS,AX
[V83] 3052: ;Zielstringsegment
[G73] 3053: MOV ES,DS:[SI+256]
[3F1] 3054: MOV DI,1
[WAZ] 3055: ;Stringlänge
[UK2] 3056: MOV BL,ES:[0]
[VE1] 3057: MOV CH,0
[ZE1] 3058: MOV CL,0
[U94] 3059: ;Länge des Quellstrings
[C90] 3060: LODSB
[B81] 3061: CMP AL,BL
[NDI] 3062: JAE X430
[UPT] 3063: ;Anzahl der führenden Leerzeichen berechnen
[SU1] 3064: MOV CL,BL
[G81] 3065: SUB CL,BL
[YT1] 3066: MOV BL,AL
[WU3] 3067: ;Leerzeichen schreiben
[M02] 3068: X430: MOV AL," "
[KX1] 3069: REP STOSB
[SU1] 3070: MOV CL,BL
[UN3] 3071: ;Zeichen übertragen
[X11] 3072: REP MOVSB
[6W1] 3073: MOV DS,DX
[G71] 3074: RET 258
[000] 3075:
[AP1] 3076: ;PUT*
[LE2] 3077: L44: MOV BP,SP
[L32] 3078: ;Datennummer
[Q23] 3079: MOV BX,SS:[BP+6]
[K24] 3080: ;Adresse des Feldarrays
[TD1] 3081: SHL BX,1
[N93] 3082: MOV SI,DS:[28+BX]
[HE1] 3083: SHR BX,1
[YX3] 3084: ;Länge des Datensatzes
[R73] 3085: MOV AL,DS:[26+BX]
[LE1] 3086: MOV AH,0
[F91] 3087: PUSH AX
[KM1] 3088: ;Position
[Q23] 3089: MOV DX,SS:[BP+2]
[Q23] 3090: MOV CX,SS:[BP+4]
[Y24] 3091: ;Ins Long-Integer-Format
[WH1] 3092: OR CL,00H
[E82] 3093: X443: CMP CL,02
[3E1] 3094: JAE X444
[9E1] 3095: SHR CL,1
[B81] 3096: RCR DX,1
[W41] 3097: INC CH
```



```
[GG1] 3098: JMB X443
[GG2] 3099: X444: JMB DX,1
[GG3] 3100: SBB AX,0
[GG4] 3101: MUL CL
[GG5] 3102: MOV CX,AX
[GG6] 3103: ;Mit Länge eines Datensatzes multiplizieren
[GG7] 3104: POP AX
[GG8] 3105: MOV DX
[GG9] 3106: ADD DX,CX
[GG10] 3107: MOV DX,AX
[GG11] 3108: ;Datel-Handle
[GG12] 3109: MOV BL,DS:[BX]
[GG13] 3110: ;Zeiger positionieren
[GG14] 3111: MOV AX,4200H
[GG15] 3112: INT 21H
[GG16] 3113: ;Anzahl der Felder
[GG17] 3114: LODSB
[GG18] 3115: MOV AH,0
[GG19] 3116: MOV DI,AX
[GG20] 3117: ;Länge eines Feldes
[GG21] 3118: X440: LODSW
[GG22] 3119: MOV CX,AX
[GG23] 3120: PUSH DS
[GG24] 3121: ;Segmentadresse des Strings
[GG25] 3122: LODSW
[GG26] 3123: MOV DS,AX
[GG27] 3124: MOV DX,1
[GG28] 3125: ;Zeichen in Datei schreiben
[GG29] 3126: MOV AH,40H
[GG30] 3127: INT 21H
[GG31] 3128: POP DS
[GG32] 3129: ;Fehler?
[GG33] 3130: JC X441
[GG34] 3131: CMP AX,CX
[GG35] 3132: JNE X441
[GG36] 3133: ;Nächstes Feld behandeln
[GG37] 3134: DEC DI
[GG38] 3135: JNE X440
[GG39] 3136: X441: RET 6
[GG40] 3137:
[GG41] 3138: ;Konvertierung Integer -> Single
[GG42] 3139: L45: MOV BP,SP
[GG43] 3140: ;Rücksprungadresse
[GG44] 3141: MOV DI,SS:[BP]
[GG45] 3142: ;Integer-Wert
[GG46] 3143: MOV AX,SS:[BP+2]
[GG47] 3144: MOV BL,0
[GG48] 3145: AND AX,AX
[GG49] 3146: JE X451
[GG50] 3147: MOV BX,7F90H
[GG51] 3148: ;Wert positiv?
[GG52] 3149: JNS X450
[GG53] 3150: NEG AX
[GG54] 3151: ;AND-Maske für Vorzeichen
[GG55] 3152: MOV BH,0FFH
[GG56] 3153: ;Linksbündig machen
[GG57] 3154: X450: AND AX,AX
[GG58] 3155: JS X452
[GG59] 3156: SHL AX,1
[GG60] 3157: ;Exponent mitzählen
[GG61] 3158: DEC BL
[GG62] 3159: JNC X450
[GG63] 3160: ;Vorzeichen bestimmen
[GG64] 3161: X452: AND AH,BH
[GG65] 3162: ;Ergebnis auf Stack
[GG66] 3163: X451: MOV BYTE PTR SS:[BP],0
[GG67] 3164: MOV DI,SS:[BP+1],AX
[GG68] 3165: MOV SS:[BP+1],BL
[GG69] 3166: JMP DI
[GG70] 3167:
[GG71] 3168: ;Konvertierung Single -> String (STR$)
[GG72] 3169: L46: SUB SP,250
[GG73] 3170: ;Platz für String schaffen
[GG74] 3171: MOV DI,SP
[GG75] 3172: ;Single-Wert lesen
[GG76] 3173: MOV BL,0
[GG77] 3174: MOV DH,SS:[DI+252]
[GG78] 3175: MOV SI,SS:[DI+253]
[GG79] 3176: MOV BL,SS:[DI+255]
[GG80] 3177: MOV AX,SS
[GG81] 3178: MOV ES,AX
[GG82] 3179: INC DI
[GG83] 3180: ;Erstes Zeichen je nach Vorzeichen bestimmen
[GG84] 3181: MOV AL," "
[GG85] 3182: AND SI,SI
[GG86] 3183: JNS X460
[GG87] 3184: MOV AL,"-"
[GG88] 3185: X460: STOSB
[GG89] 3186: ;Ist die Zahl Null?
[GG90] 3187: CMP BL,0
[GG91] 3188: JNE X462
[GG92] 3189: MOV BL,0
[GG93] 3190: STOSB
[GG94] 3191: ;Stringlänge setzen, Routine verlassen
[GG95] 3192: X461: MOV BP,SP
[GG96] 3193: LEA AX,[DI-1]
[GG97] 3194: SUB AX,BP
[GG98] 3195: MOV SS:[BP],AL
[GG99] 3196: ;String mit Leerzeichen abschließen (für PR
[GG100] 3197: MOV AL," "
[GG101] 3198: STOSB
[GG102] 3199: JMP SS:[BP+250]
[GG103] 3200: ;Zahl ins Binär-Format
[GG104] 3201: X462: STOSB
[GG105] 3202: XOR CX,CX
[GG106] 3203: ;Zahl < 1?
[GG107] 3204: CMP BL,129
[GG108] 3205: JB X463
[GG109] 3206: JMP X46L
[GG110] 3207: ;Zahl = Zahl x 10
[GG111] 3208: X463: CALL X46C
[GG112] 3209: ;10er-Exponent um 1 verringern
[GG113] 3210: DEC CL
[GG114] 3211: ;Zahl immer noch < 1?
[GG115] 3212: CMP BL,128
[GG116] 3213: JB X463
[GG117] 3214: ;Mantisse runden
[GG118] 3215: ADD DX,128
[GG119] 3216: ADC SI,0
[GG120] 3217: JNC X464
[GG121] 3218: RCR SI,1
[GG122] 3219: RCR DX,1
[GG123] 3220: INC BL
[GG124] 3221: X464: CMP BL,129
[GG125] 3222: JAE X465
[GG126] 3223: CALL X46C
[GG127] 3224: DEC CL
[GG128] 3225: ;Sieben Mantissen-Ziffern ausgeben
[GG129] 3226: X465: CALL X46G
[GG130] 3227: SUB CH,CL
[GG131] 3228: ;Ist Exponentenformat nötig?
[GG132] 3229: CMP CH,7
[GG133] 3230: JA X46A
[GG134] 3231: ;Dezimalpunkt dann nach links
[GG135] 3232: MOV BP,SP
[GG136] 3233: MOV AX,SS:[BP+2]
[GG137] 3234: XCHG AL,AH
[GG138] 3235: MOV SS:[BP+2],AX
[GG139] 3236: ADD BP,3
[GG140] 3237: CMP DI,BP
[GG141] 3238: JNE X466
[GG142] 3239: INC DI
[GG143] 3240: X466: MOV SI,DI
[GG144] 3241: NOT CL
[GG145] 3242: MOV CH,0
[GG146] 3243: ADD DI,CX
[GG147] 3244: PUSH
[GG148] 3245: ;Ziffernfolge um die Anzahl führender Nullen
[GG149] 3246: nach rechts schieben
```

```
[322] 3246: X467: DEC SI
[323] 3247: DEC DI
[324] 3248: MOV AX,SS:[SI]
[325] 3249: MOV SS:[DI],AL
[326] 3250: CMP SI,BP
[327] 3251: JA X467
[328] 3252: JCXZ X469
[329] 3253: ;Führende Nullen schreiben
[330] 3254: X468: DEC DI
[331] 3255: MOV BYTE PTR SS:[DI],0
[332] 3256: LOOP X468
[333] 3257: X469: POP DI
[334] 3258: JMP X461
[335] 3259: ;Exponent aus DI in String schreiben
[336] 3260: X46A: MOV AL,"E"
[337] 3261: STOSB
[338] 3262: ;Vorzeichen des Exponenten
[339] 3263: MOV AL,"-"
[340] 3264: AND CL,CL
[341] 3265: JNS X46B
[342] 3266: NEG CL
[343] 3267: MOV AL,"-"
[344] 3268: X46B: STOSB
[345] 3269: ;Exponent ins ASCII-Format
[346] 3270: MOV AH,0
[347] 3271: MOV AH,0
[348] 3272: MOV CL,10
[349] 3273: DIV CL
[350] 3274: OR AX,00H
[351] 3275: STOSW
[352] 3276: JMP X461
[353] 3277: ;Zahl mit 10 multiplizieren
[354] 3278: X46C: MOV AX,DX
[355] 3279: MOV BP,SI
[356] 3280: ;X x 10 = (X x 4 + X) x 2
[357] 3281: SHR BP,1
[358] 3282: RCR AX,1
[359] 3283: BP,1
[360] 3284: RCR AX,1
[361] 3285: ADC DX,AX
[362] 3286: ADC SI,BP
[363] 3287: ;Überlauf aufgetreten?
[364] 3288: JNC X46D
[365] 3289: RCR SI,1
[366] 3290: RCR DX,1
[367] 3291: INC BP
[368] 3292: ;Exponent angleichen
[369] 3293: X46D: ADD BL,3
[370] 3294: RET
[371] 3295: X46E: MOV AL,0
[372] 3296: X46F: RCL DX,1
[373] 3297: RCL SI,1
[374] 3298: RCL AL,1
[375] 3299: DEC BL
[376] 3300: CMP BL,128
[377] 3301: JA X46F
[378] 3302: OR AL,0
[379] 3303: STOSB
[380] 3304: RET
[381] 3305: ;Mantisse mit Dezimalpunkt ausgeben
[382] 3306: X46G: CALL X46E
[383] 3307: MOV AL,"."
[384] 3308: STOSB
[385] 3309: X46H: CALL X46C
[386] 3310: CALL X46E
[387] 3311: INC CH
[388] 3312: CMP CH,6
[389] 3313: JB X46H
[390] 3314: ;Nachfolgende Nullen entfernen
[391] 3315: X46I: MOV AL,ES:[DI-1]
[392] 3316: DEC AL,0
[393] 3317: X46J: MOV AL,0
[394] 3318: DEC CH
[395] 3319: X46J: DEC DI
[396] 3320: JMP X46I
[397] 3321: X46K: CMP AL,"."
[398] 3322: JE X46J
[399] 3323: RET
[400] 3324: ;Behandlung für Zahlen > 1
[401] 3325: X46L: CALL X46G
[402] 3326: ;Durch 10 dividieren, Exponent mitzählen
[403] 3327: INC CL
[404] 3328: CMP BL,129
[405] 3329: JAE X46L
[406] 3330: DEC CL
[407] 3331: ADD DX,128
[408] 3332: ADC SI,0
[409] 3333: JNC X46M
[410] 3334: RCR SI,1
[411] 3335: RCR DX,1
[412] 3336: INC BL
[413] 3337: X46M: CALL X46C
[414] 3338: CALL X46G
[415] 3339: ;Exponentendarstellung notwendig?
[416] 3340: CMP CL,7
[417] 3341: JB X46N
[418] 3342: JMP X46A
[419] 3343: X46N: MOV BP,SP
[420] 3344: ADD BP,3
[421] 3345: ;Dezimalpunkt verschieben
[422] 3346: X46O: DEC CL
[423] 3347: JS X46P
[424] 3348: MOV AX,SS:[BP]
[425] 3349: XCHG AL,AH
[426] 3350: MOV SS:[BP],AX
[427] 3351: INC BP
[428] 3352: JMP X46O
[429] 3353: ;Eventuell entfernte Nullen wiederherstellen
[430] 3354: X46P: CMP DI,BP
[431] 3355: JAE X46Q
[432] 3356: MOV DI,BP
[433] 3357: ;Wenn ganze Zahl, dann Dezimalpunkt entfernen
[434] 3358: X46Q: CMP BYTE PTR SS:[DI-1],0
[435] 3359: JNE X46R
[436] 3360: DEC DI
[437] 3361: X46R: JMP X46I
[438] 3362: ;Zahl durch 10 dividieren
[439] 3363: X46S: PUSH DX
[440] 3364: MOV BP,10
[441] 3365: MOV AX,SI
[442] 3366: XOR DX,DX
[443] 3367: DIV BP
[444] 3368: MOV SI,AX
[445] 3369: POP AX
[446] 3370: DIV BP
[447] 3371: XCHG AX,DX
[448] 3372: MOV AH,26
[449] 3373: MUL AH
[450] 3374: ;Zahl linksbündig machen
[451] 3375: X46T: SHL AL,1
[452] 3376: RCL DX,1
[453] 3377: RCL SI,1
[454] 3378: DEC BL
[455] 3379: AND SI,SI
[456] 3380: JNS X46T
[457] 3381: RET
[458] 3382:
[459] 3383: ;Single negieren (Negatives Vorzeichen)
[460] 3384: L47: MOV BP,SP
[461] 3385: ;Vorzeichen umdrehen
[462] 3386: XOR BYTE PTR SS:[BP+4],128
[463] 3387: RET
[464] 3388:
[465] 3389: ;Single ***
[466] 3390: L48: MOV DX,ADDS-L00
[467] 3391: JMP DX
[468] 3392:
[469] 3393: ;Konvertierung Integer -> Single (zweiter
[470] 3394: ;Wert, für Operationen
[471] 3395: L49: SUB SP,2
[472] 3396: MOV BP,SP
[473] 3397: ;Platz für Single schaffen
[474] 3398: XCHG AX,SS:[BP+6]
[475] 3399: XCHG AX,SS:[BP+2]
[476] 3400: MOV SS:[BP],AX
[477] 3401: MOV AX,SS:[BP+8]
[478] 3402: MOV BL,0
[479] 3403: AND AX,AX
[480] 3404: JE X491
[481] 3405: MOV BX,7F90H
[482] 3406: JNS X490
[483] 3407: NEG AX
[484] 3408: MOV BH,0FFH
[485] 3409: AND AX,AX
[486] 3410: JS X492
[487] 3411: SHL AX,1
[488] 3412: DEC BL
[489] 3413: JMP X490
[490] 3414: X492: AND AH,BH
[491] 3415: MOV BYTE PTR SS:[BP+6],0
[492] 3416: MOV SS:[BP+7],AX
[493] 3417: MOV SS:[BP+9],BL
[494] 3418: RET
[495] 3419:
[496] 3420: ;Single **
[497] 3421: L4A: MOV DX,MULS-L00
[498] 3422: L4A: MOV DX
[499] 3423:
[500] 3424: ;Single */
[501] 3425: L4B: MOV DX,DIVS-L00
[502] 3426: JMP DX
[503] 3427:
[504] 3428: ;Funktion *LOG
[505] 3429: L4C: MOV DX,200J-L00
[506] 3430: L4C: MOV DX,200J-L00
[507] 3431: JMP DX
[508] 3432:
[509] 3433: ;Single **, Potenz A ^ B = EXP (LOG (A) x
[510] 3434: B)
[511] 3435: L4D: MOV BP,SP
[512] 3436: MOV AX,SS:[BP+8]
[513] 3437: AND AX,AX
[514] 3438: JE X4D0
[515] 3439: ;Vorzeichen retten
[516] 3440: PUSH AX
[517] 3441: ;Absolutwert von A auf Stack
[518] 3442: AND AL,7FH
[519] 3443: PUSH AX
[520] 3444: PUSH SS:[BP+6]
[521] 3445: ;Logarithmus berechnen
[522] 3446: MOV DX,200J-L00
[523] 3447: CALL DX
[524] 3448: ;A auf Stack
[525] 3449: MOV BP,SP
[526] 3450: PUSH SS:[BP+10]
[527] 3451: PUSH SS:[BP+8]
[528] 3452: ;A x B berechnen
[529] 3453: MOV DX,MULS-L00
[530] 3454: CALL DX
[531] 3455: ;Exponentialfunktion
[532] 3456: MOV DX,200E-L00
[533] 3457: CALL DX
[534] 3458: ;Vorzeichen in das Ergebnis holen
[535] 3459: POP AX
[536] 3460: POP BX
[537] 3461: POP CX
[538] 3462: MOV BL,80H
[539] 3463: OR BL,CL
[540] 3464: ;Ergebnis auf Stack
[541] 3465: MOV BP,SP
[542] 3466: MOV SS:[BP+6],AX
[543] 3467: MOV SS:[BP+8],BX
[544] 3468: X4D0: RET 4
[545] 3469:
[546] 3470: ;ON X GOSUB
[547] 3471: L4E: MOV BP,SP
[548] 3472: ;Wert holen
[549] 3473: MOV BX,SS:[BP+4]
[550] 3474: ;Null ist ungültig
[551] 3475: JS X4E0
[552] 3476: JS X4E0
[553] 3477: ;Zeiger auf Tabelle berechnen
[554] 3478: SHL BX,1
[555] 3479: ADD BX,SS:[BP+2]
[556] 3480: ;Grenze der Tabelle berechnen
[557] 3481: MOV DX,SS:[BP]
[558] 3482: SUB DX,3
[559] 3483: ;Außerhalb der Tabelle?
[560] 3484: CMP BX,DX
[561] 3485: JAE X4E0
[562] 3486: ;Zum Tabelleneintrag springen
[563] 3487: MOV AX,CS:[BX]
[564] 3488: MOV SS:[BP],AX
[565] 3489: X4E0: RET 4
[566] 3490:
[567] 3491: ;ON X GOSUB
[568] 3492: L4F: MOV BP,SP
[569] 3493: ;X holen
[570] 3494: MOV BX,SS:[BP+4]
[571] 3495: ;0 nicht erlaubt
[572] 3496: DEC BX
[573] 3497: JS X4E0
[574] 3498: ;Tabelleneintrag holen
[575] 3499: SHL BX,1
[576] 3500: ADD BX,SS:[BP+2]
[577] 3501: MOV DX,SS:[BP]
[578] 3502: SUB DX,3
[579] 3503: ;Außerhalb, dann zurück
[580] 3504: CMP BX,DX
[581] 3505: JAE X4E0
[582] 3506: MOV AX,CS:[BX]
[583] 3507: ;Rücksprungadresse auf Stack (erste Instruk
[584] 3508: tion hinter der Tabelle)
[585] 3509: XCHG AX,SS:[BP]
[586] 3510: ;Zur Unterroutine springen
[587] 3511: MOV SS:[BP+4],AX
[588] 3512: RET 2
[589] 3513: X4F0: RET 4
[590] 3514: ;Funktion *INT*
[591] 3515: L50: MOV DX,1000-L00
[592] 3516: JMP DX
[593] 3517:
[594] 3518: ;Konvertierung Single -> Integer (mit Rundung)
[595] 3519: L51: MOV BP,SP
[596] 3520: MOV AX,SS:[BP+3]
[597] 3521: MOV CH,SS:[BP+5]
[598] 3522: ;Wert gleich Null?
[599] 3523: AND CH,CH
[600] 3524: JE X510
[601] 3525: ;Vorzeichen retten
[602] 3526: MOV BL,AH
[603] 3527: ;Ins Binär-Format
[604] 3528: OR AH,80H
[605] 3529: MOV CL,144
[606] 3530: ;Überlauf?
[607] 3531: SUB CL,CH
[608] 3532: JRC X510
[609] 3533: SHR AX,CL
```

Listing 2. Der Quelltext der SBasic-Laufzeitbibliothek »sbasic.asm«


```
[XCL] 3534: ;Runden
[YA1] 3535: ADC AX,0
[HD4] 3536: ;Vorzeichen zurückholen
[YP1] 3537: AND BL,BL
[HG1] 3538: JNS X510
[351] 3539: NEG AX
[783] 3540: ;Ergebnis auf Stack
[AR3] 3541: MOV SS:[BP+4],AX
[821] 3542: RET 2
[000] 3543:
[554] 3544: ;Single Vergleich ==
[K12] 3545: L52: POP ES
[KA9] 3546: ;Rücksprungadresse sichern, zwei Zahlen vor
gleichen
[6K2] 3547: MOV DX,2000-L00
[D61] 3548: CALL DX
[B36] 3549: ;Vergleichsergebnis initialisieren
[8V1] 3550: XOR AX,AX
[BR7] 3551: ;Rückgabewert der Vergleichsroutine testen
[ND1] 3552: CMP DL,1
[VE1] 3553: JNE X520
[YA4] 3554: ;true zurückgeben
[Z31] 3555: DEC AX
[PR3] 3556: X520: MOV SS:[BP+8],AX
[U91] 3557: PUSH ES
[V21] 3558: RET 6
[000] 3559:
[312] 3560: ;Single Vergleich <=
[P12] 3561: L53: POP ES
[6K2] 3562: MOV DX,2000-L00
[D61] 3563: CALL DX
[8V1] 3564: XOR AX,AX
[ND1] 3565: CMP DL,-1
[XE1] 3566: JNE X530
[Z31] 3567: DEC AX
[UR3] 3568: X530: MOV SS:[BP+8],AX
[U91] 3569: PUSH ES
[V21] 3570: RET 6
[000] 3571:
[454] 3572: ;Single Vergleich ==
[U12] 3573: L54: POP ES
[6K2] 3574: MOV DX,2000-L00
[D61] 3575: CALL DX
[8V1] 3576: XOR AX,AX
[ND1] 3577: CMP DL,-1
[Z31] 3578: JNE X540
[Z31] 3579: DEC AX
[YR3] 3580: X540: MOV SS:[BP+8],AX
[U91] 3581: PUSH ES
[V21] 3582: RET 6
[000] 3583:
[DE4] 3584: ;Single Vergleich >=
[U12] 3585: L55: POP ES
[6K2] 3586: MOV DX,2000-L00
[D61] 3587: CALL DX
[8V1] 3588: XOR AX,AX
[ND1] 3589: CMP DL,-1
[Z31] 3590: JNE X550
[Z31] 3591: DEC AX
[UR3] 3592: X550: MOV SS:[BP+8],AX
[U91] 3593: PUSH ES
[V21] 3594: RET 6
[000] 3595:
[DE4] 3596: ;Single Vergleich <=
[U12] 3597: L56: POP ES
[6K2] 3598: MOV DX,2000-L00
[D61] 3599: CALL DX
[8V1] 3600: XOR AX,AX
[ND1] 3601: CMP DL,1
[XE1] 3602: JNE X560
[Z31] 3603: DEC AX
[AS3] 3604: X560: MOV SS:[BP+8],AX
[U91] 3605: PUSH ES
[V21] 3606: RET 6
[000] 3607:
[000] 3608:
[DE4] 3609: ;Single Vergleich <=
[U12] 3610: L57: POP ES
[6K2] 3611: MOV DX,2000-L00
[D61] 3612: CALL DX
[8V1] 3613: XOR AX,AX
[ND1] 3614: CMP DL,0
[XE1] 3615: JNE X570
[Z31] 3616: DEC AX
[AS3] 3617: X570: MOV SS:[BP+8],AX
[U91] 3618: PUSH ES
[V21] 3619: RET 6
[000] 3620:
[UR4] 3621: ;Rücksprung nach FN1
[6G2] 3622: L58: MOV BP,SP
[YPA] 3623: ;Rücksprungadresse holen, Ergebnis um 2 Byt
e nach oben schieben
[M13] 3624: MOV AX,SS:[BP+2]
[T03] 3625: XCHG AX,SS:[BP+4]
[X03] 3626: XCHG AX,SS:[BP+6]
[86] 3627: ;Zwei Rücksprungadressen vergessen
[ED1] 3628: ADD SP,4
[R61] 3629: JMP AX
[000] 3630:
[M04] 3631: ;Funktion *SGN* (Single)
[AG2] 3632: L59: MOV BP,SP
[U74] 3633: ;Ergebnis initialisieren
[HW1] 3634: XOR DX,DX
[S13] 3635: MOV AX,SS:[BP+4]
[T11] 3636: ;Zahl = 0?
[DI1] 3637: CMP AH,0
[U91] 3638: JE X590
[G51] 3639: INC DX
[U22] 3640: ;Zahl positiv?
[R11] 3641: AND AL,AL
[LH1] 3642: JNS X520
[F51] 3643: NEG DX
[0L4] 3644: ;Ergebnis zurückschreiben
[D33] 3645: X590: MOV SS:[BP+4],DX
[B21] 3646: RET 2
[000] 3647:
[XU4] 3648: ;Ausgabe-Funktion *TAB*
[AN2] 3649: L5A: MOV BP,SP
[S13] 3650: ;Positionswert holen
[G13] 3651: MOV CL,SS:[BP+2]
[S11] 3652: AND CL,CL
[AL1] 3653: JE X5A0
[V31] 3654: DEC CL
[743] 3655: ;Dateinummer holen
[6R3] 3656: X5A0: MOV BX,SS:[BP+4]
[Z32] 3657: ;Bildschirm?
[ZV3] 3658: CMP BYTE PTR DS:[BX],-1
[GAL] 3659:
[UTB] 3660: ;Neue Position kleiner als momentane Positi
on, dann in neue Zeile gehen
[622] 3661: CMP DS:[4+BX],CL
[B91] 3662: JBE X5A1
[P13] 3663: PUSH CX
[6N1] 3664: MOV AL,13
[AG2] 3665: MOV DX,X006-L00
[D61] 3666: CALL DX
[U91] 3667: MOV AL,10
[AG2] 3668: MOV DX,X006-L00
[D61] 3669: CALL DX
[H71] 3670: POP CX
[XU9] 3671: ;Bis zur angegebenen Position mit Leerzeich
en auffüllen
[G13] 3672: X5A1: CMP DS:[4+BX],CL
[L11] 3673: JAE X5A5
[R91] 3674: PUSH CX
[FK1] 3675: MOV AL,-
[AG2] 3676: MOV DX,X006-L00
[D61] 3677: CALL DX
[H71] 3678: POP CX
[S11] 3679: JMP X5A1
```

```
[YW3] 3680: ;Bildschirmseite holen
[752] 3681: X5A2: MOV AH,0FH
[G81] 3682: INT 10H
[029] 3683: ;Tabulator-Position größer als verfügbare Z
eilenlänge?
[W22] 3684: X5A3: SUB CL,AH
[V21] 3685: JAE X5A3
[HN1] 3686: ADD CL,AH
[R91] 3687: PUSH CX
[AM3] 3688: ;Cursorposition lesen
[7R1] 3689: MOV AH,03H
[G81] 3690: INT 10H
[H71] 3691: POP CX
[R91] 3692: PUSH CX
[S57] 3693: ;Position kleiner, dann in neue Zeile gehen
[U51] 3694: CMP DL,CL
[L11] 3695: JBE X5A4
[3W1] 3696: XOR BX,BX
[6N1] 3697: MOV AL,13
[AG2] 3698: MOV DX,X006-L00
[D61] 3699: CALL DX
[M21] 3700: ;Seite lesen
[952] 3701: X5A4: MOV AH,0FH
[G81] 3702: INT 10H
[M52] 3703: ;Position lesen
[7R1] 3704: MOV AH,03H
[G81] 3705: INT 10H
[H71] 3706: POP CX
[D44] 3707: ;Neue Position erreicht?
[U51] 3708: CMP DL,CL
[L11] 3709: JAE X5A5
[R91] 3710: PUSH CX
[L74] 3711: ;Ein Leerzeichen ausgeben
[3W1] 3712: XOR BX,BX
[FK1] 3713: MOV AL,-
[AG2] 3714: MOV DX,X006-L00
[D61] 3715: CALL DX
[E91] 3716: JMP X5A4
[AT1] 3717: X5A5: RET 2
[000] 3718:
[AS4] 3719: ;Print-Funktion *SPC*
[R91] 3720: L5B: MOV BP,SP
[P15] 3721: ;Anzahl der Leerzeichen lesen
[W13] 3722: MOV CX,SS:[BP+2]
[TD1] 3723: CMP CX,0
[U21] 3724: JBE X5B1
[743] 3725: ;Dateinummer holen
[W13] 3726: MOV BX,SS:[BP+4]
[R82] 3727: X5B0: PUSH CX
[K13] 3728: ;Leerzeichen ausgeben
[FK1] 3729: MOV AL,-
[AG2] 3730: MOV DX,X006-L00
[D61] 3731: CALL DX
[H71] 3732: POP CX
[U91] 3733: LOOP X5B0
[AT1] 3734: X5B1: RET 2
[000] 3735:
[HBD] 3736: ;Cursorposition Y lesen (*LOCATE* mit fehle
nden Parametern und *CRLIN*)
[KH2] 3737: L5C: MOV BP,SP
[YW3] 3738: ;Bildschirmseite holen
[L71] 3739: MOV AH,0FH
[G81] 3740: INT 10H
[L13] 3741: ;Position bestimmen
[7R1] 3742: MOV AH,03H
[G81] 3743: INT 10H
[ET1] 3744: MOV DL,DH
[Z11] 3745: MOV DH,0
[TD1] 3746: ;1 addieren
[G51] 3747: INC DX
[0L4] 3748: XCHG DX,SS:[BP]
[SP2] 3749: JMP DX
[371] 3750:
[000] 3751:
[CF8] 3752: ;Cursorposition X lesen (*LOCATE* und *POS*)
[PH2] 3753: L5D: MOV BP,SP
[L71] 3754: MOV AH,0FH
[G81] 3755: INT 10H
[7R1] 3756: MOV AH,03H
[G81] 3757: INT 10H
[Z11] 3758: MOV DH,0
[G51] 3759: INC DX
[SP2] 3760: XCHG DX,SS:[BP]
[371] 3761: JMP DX
[000] 3762:
[W85] 3763: ;Cursorform setzen (*LOCATE*)
[UH2] 3764: L5E: MOV BP,SP
[QW3] 3765: ;Untere Cursorzeile
[G13] 3766: MOV CL,SS:[BP+2]
[R43] 3767: ;Obere Cursorzeile
[G13] 3768: MOV CH,SS:[BP+4]
[VL3] 3769: ;Cursorform sichern
[RH2] 3770: MOV DS:[36],CX
[KR3] 3771: ;Cursor momentan aus?
[904] 3772: CMP WORD PTR SS:[BP+6],0
[L01] 3773: JNE X5E0
[XW4] 3774: ;Cursor unsichtbar schalten
[E32] 3775: MOV AX,1F00H
[PP2] 3776: X5E0: MOV AH,01H
[G81] 3777: INT 10H
[V21] 3778: RET 6
[000] 3779:
[U77] 3780: ;Cursorzustand ein/aus holen (*LOCATE*)
[YH2] 3781: L5F: MOV BP,SP
[ME1] 3782: ;Seite lesen
[L71] 3783: MOV AH,0FH
[G81] 3784: INT 10H
[952] 3785: ;Position holen
[7R1] 3786: MOV AH,03H
[G81] 3787: INT 10H
[U84] 3788: ;Mit Aus-Form vergleichen
[W12] 3789: SUB CX,1F00H
[MP2] 3790: XCHG CX,SS:[BP]
[Z61] 3791: JMP CX
[000] 3792:
[U96] 3793: ;Obere Cursorzeile holen (*LOCATE*)
[9F2] 3794: L60: MOV BP,SP
[LH2] 3795: MOV AL,DS:[37]
[L21] 3796: MOV AH,0
[783] 3797: ;Ergebnis auf Stack
[CP2] 3798: XCHG AX,SS:[BP]
[R61] 3799: JMP AX
[000] 3800:
[776] 3801: ;Untere Cursorzeile holen (*LOCATE*)
[DF2] 3802: L61: MOV BP,SP
[XG2] 3803: MOV AL,DS:[36]
[L21] 3804: MOV AH,0
[CP2] 3805: XCHG AX,SS:[BP]
[R61] 3806: JMP AX
[000] 3807:
[F94] 3808: ;Vordergrundfarbe holen (*COLOR* mit fehle
nden Parametern)
[HF2] 3809: L62: MOV BP,SP
[EL2] 3810: MOV AL,DS:[9]
[LH2] 3811: ;Blink-Bit sichern
[YT1] 3812: MOV AL,0
[CK4] 3813: ;Hintergrundfarbe ausblenden
[RK1] 3814: AND AX,15
[YK2] 3815: ;Blink-Bit dazu
[BP1] 3816: MOV CL,4
[BT1] 3817: SHR BL,CL
[LK1] 3818: AND BL,16
[KL1] 3819: OR AL,BL
[RV3] 3820: ;Ergebnis zurückgeben
[CP2] 3821: XCHG AX,SS:[BP]
[R61] 3822: JMP AX
[000] 3823:
[V26] 3824: ;Hintergrundfarbe holen (*COLOR*)
[NF2] 3825: L63: MOV BP,SP
[EL2] 3826: MOV AL,DS:[9]
```

```
[KN4] 3827: ;Hintergrundfarbe in Bit 0..2
[BF1] 3828: MOV CL,4
[771] 3829: SHR AL,CL
[NCL] 3830: AND AX,7
[783] 3831: ;Ergebnis auf Stack
[CP2] 3832: XCHG AX,SS:[BP]
[R61] 3833: JMP AX
[000] 3834:
[R57] 3835: ;Konvertierung String -> Single (*VAL*)
[F72] 3836: L64: PUSH DS
[SW1] 3837: MOV AX,SS
[XV1] 3838: MOV AX,DX
[571] 3839: MOV SI,SP
[7D1] 3840: ADD SI,4
[G83] 3841: ;Stringlänge holen
[CV3] 3842: LODSB
[ZU1] 3843: MOV CL,AL
[VE1] 3844: MOV CH,0
[AA4] 3845: ;Zahl mit 0 initialisieren
[3W1] 3846: XOR BX,BX
[HW1] 3847: XOR DX,DX
[VU1] 3848: XOR BP,BP
[ZR4] 3849: ;Stringlänge = 0, dann fertig
[HK1] 3850: JCXZ X644
[4U6] 3851: ;Unterprogramm "Lies nächstes Zeichen"
[VC2] 3852: X640: CALL X647
[X91] 3853: JE X644
[962] 3854: ;Vorzeichen?
[9K1] 3855: CMP AL,-
[H91] 3856: JE X640
[HK1] 3857: CMP AL,-
[3F1] 3858: JNE X641
[H71] 3859: NOT BP
[9C1] 3860: JMP X640
[MY6] 3861: ;Oktalzahlen werden nicht unterstützt!
[HD2] 3862: X641: CMP AL,-
[SF1] 3863: JNE X648
[ZK1] 3864: CALL X647
[X91] 3865: JE X644
[UAS] 3866: ;Ist es eine hexadezimale Zahl?
[XN1] 3867: CMP AL,"h"
[R91] 3868: JE X642
[XN1] 3869: CMP AL,"H"
[CF1] 3870: JNE X644
[N03] 3871: ;Eine Ziffer lesen
[XC2] 3872: X642: CALL X647
[X91] 3873: JE X644
[W84] 3874: ;Ins Binärformat wandeln
[XK1] 3875: SUB AL,"0"
[H91] 3876: JE X644
[ZK2] 3877: ;Dezimalziffer?
[8L1] 3878: CMP AL,10
[H91] 3879: JB X643
[U54] 3880: ;Großbuchstabe A bis F?
[UD1] 3881: SUB AL,40
[8L1] 3882: CMP AL,10
[H91] 3883: JB X644
[QM1] 3884: CMP AL,"a"
[H91] 3885: JB X643
[FX3] 3886: ;Kleinbuchstabe a bis f?
[PL1] 3887: SUB AL,32
[M91] 3888: JB X644
[QM1] 3889: CMP AL,16
[5E1] 3890: JAE X644
[683] 3891: ;Vorherigen Wert mal 16
[A82] 3892: X643: SHL DX,1
[1E1] 3893: SHL DX,1
[1E1] 3894: SHL DX,1
[1E1] 3895: SHL DX,1
[7L7] 3896: ;Neue Ziffer dazuzählen, nächstes Zeichen
[SP1] 3897: ADD DL,AL
[F91] 3898: JMP X642
[VU6] 3899: ;Integer-Zahl ins Single-Format wandeln
[VC2] 3900: X644: AND DX,DX
[6A1] 3901: JE X647
[ZR1] 3902: MOV BL,144
[FX3] 3903: ;Zahl größer 32767?
[7H1] 3904: JNS X645
[F51] 3905: NEG DX
[H71] 3906: NOT BP
[N23] 3907: ;Linksbündig machen
[CD2] 3908: AND DH,DH
[DB1] 3909: JS X646
[1E1] 3910: SHL DX,1
[773] 3911: ;Exponent mitzählen
[R31] 3912: DEC BL
[SG1] 3913: JMP X645
[Z14] 3914: ;Zahl ins Single-Format
[DN2] 3915: X646: AND DH,7FH
[ZK4] 3916: ;Vorzeichen-Bit einsetzen
[KR1] 3917: AND BP,BP
[6A1] 3918: JE X647
[MH1] 3919: OR DH,80H
[RV3] 3920: ;Ergebnis zurückgeben
[ZK2] 3921: X647: MOV BP,SP
[Z43] 3922: MOV SS:[BP+256],BH
[553] 3923: MOV SS:[BP+257],DX
[653] 3924: MOV SS:[BP+259],BL
[G71] 3925: POP DS
[471] 3926: RET 252
[V35] 3927: ;Behandlung von Dezimalzahlen
[792] 3928: X648: PUSH BP
[S55] 3929: ;Dezimalpunkt-Flag initialisieren
[GU1] 3930: MOV DI,DI
[HE2] 3931: X649: CMP AL,-
[SH1] 3932: JNE X64G
[XP1] 3933: AND DI,DI
[PG1] 3934: JNE X64B
[W31] 3935: DEC DI
[CD2] 3936: X64A: CALL X64T
[V21] 3937: JNE X649
[ZH5] 3938: ;Ist ein Dezimalpunkt vorhanden?
[0E2] 3939: X64B: AND DI,DI
[AB1] 3940: JE X64C
[S36] 3941: ;Ja, dann Kommazähler weiterzählen
[L51] 3942: INC DI
[L22] 3943: X64C: AND DI,DI
[GB1] 3944: JE X64E
[C91] 3945: PUSH DI
[L91] 3946: PUSH BX
[H91] 3947: PUSH DX
[B96] 3948: ;Vorzeichen des Exponenten abfragen
[RCL] 3949: JS X64D
[B94] 3950: ;Zahl mit 10 multiplizieren
[N12] 3951: MOV AX,8420H
[F91] 3952: PUSH AX
[WV1] 3953: XOR AX,AX
[F91] 3954: PUSH AX
[L22] 3955: MOV DX,MULS-L00
[D61] 3956: CALL DX
[N71] 3957: POP DX
[D71] 3958: POP BX
[671] 3959: POP DI
[W31] 3960: DEC DI
[ZK1] 3961: JMP X64C
[S14] 3962: ;Zahl durch 10 teilen (x.1)
[LX2] 3963: X64D: MOV AX,7D4CH
[F91] 3964: PUSH AX
[4D2] 3965: MOV AX,9CCDDH
[F91] 3966: PUSH AX
[L22] 3967: MOV DX,MULS-L00
[D61] 3968: CALL DX
[N71] 3969: POP DX
[D71] 3970: POP BX
[671] 3971: POP DI
[L51] 3972: INC DI
[ZK1] 3973: JMP X64C
[7L5] 3974: ;Vorzeichen in das Ergebnis holen
[W42] 3975: X64E: POP AX
[L51] 3976: AND AX,AX
[L81] 3977: JE X64F
```



```
[151] 3978: XOR BL,80H
[R3] 3979: ;Ergebnis zurückgeben
[R3] 3980: MOV SP,SP
[R5] 3981: MOV SS:[BP+256],DX
[R5] 3982: MOV SS:[BP+258],BX
[G7] 3983: POP DS
[47] 3984: RET 252
[DC] 3985: ;Folgt ein Exponent?
[4H] 3986: CMP AL,"e"
[7B] 3987: JE X64H
[HN] 3988: CMP AL,"E"
[UN] 3989: JNE X64H
[6E] 3990: AND DI,DI
[WB] 3991: JE X64I
[15] 3992: INC DI
[HN] 3993: XOR BP,BP
[4V] 3994: MOV ES,BP
[7N] 3995: ;Vorzeichen des Exponenten lesen
[ND] 3996: CALL X64T
[AB] 3997: JE X64C
[9B] 3998: CMP AL,"+"
[2B] 3999: JE X64J
[HK] 4000: CMP AL,"-"
[HN] 4001: JNE X64H
[EV] 4002: MOV AX,ES
[N7] 4003: NOT AX
[1W] 4004: MOV ES,AX
[RK] 4005: JMP X64J
[NB] 4006: ;Eine Ziffer des Exponenten lesen
[2E] 4007: X64T: SUB AL,"0"
[WB] 4008: JB X64I
[8L] 4009: CMP AL,"0"
[DG] 4010: JAE X64L
[UT] 4011: XCHG AX,BP
[M8] 4012: ;Alter Exponent x 10
[PM] 4013: MOV AH,10
[N6] 4014: MUL AH
[9B] 4015: ;Neue Ziffer dazuzählen
[HN] 4016: ADD BP,AX
[2K] 4017: CALL X64T
[HN] 4018: JNE X64K
[RK] 4019: X64L: MOV AX,BP
[80] 4020: CBW
[AV] 4021: MOV BP,ES
[57] 4022: ;Vorzeichen in Exponent schreiben
[RK] 4023: AND BP,BP
[PL] 4024: JNS X64H
[35] 4025: NEG AX
[HP] 4026: ;Kommazeichler aktualisieren
[YE] 4027: X64M: ADD DI,AX
[2K] 4028: SUB DI,AX
[YD] 4029: ;Auf Dezimalziffer prüfen
[0B] 4030: X64N: MOV AH,0
[XK] 4031: SUB AL,"0"
[7G] 4032: JNC X64O
[7G] 4033: ;Ungültiges Zeichen, Umwandlung beendet
[HB] 4034: X64O: JMP X64B
[K4] 4035: X64P: JE X64R
[8L] 4036: CMP AL,"0"
[PG] 4037: JAE X64O
[RK] 4038: ;Neue Ziffer ins Single-Format wandeln
[SP] 4039: MOV AH,136
[12] 4040: X64Q: DEC AH
[9D] 4041: SHL AL,1
[VL] 4042: JNS X64Q
[9R] 4043: AND AL,7FH
[BP] 4044: ;Zahl mit 10 multiplizieren
[4B] 4045: X64R: PUSH AX
[8B] 4046: PUSH SI
[C9] 4047: PUSH DI
[F9] 4048: PUSH AX
[WN] 4049: XOR AX,AX
[F9] 4050: PUSH AX
[N1] 4051: MOV AX,8420H
[F9] 4052: PUSH AX
[WN] 4053: XOR AX,AX
[F9] 4054: PUSH AX
[L9] 4055: PUSH BX
[V9] 4056: PUSH DX
[L2] 4057: MOV DX,MULS-L00
[D6] 4058: CALL DX
[9G] 4059: ;Neue Ziffer addieren
[3L] 4060: MOV DX,ADDS-L00
[D6] 4061: CALL DX
[N7] 4062: POP DX
[D7] 4063: POP BX
[D7] 4064: POP SI
[29] 4065: POP SI
[N7] 4066: POP CX
[V5] 4067: ;Kommazeichler bei Bedarf angleichen
[29] 4068: AND DI,DI
[UC] 4069: JE X64S
[W3] 4070: DEC DI
[L8] 4071: X64S: JMP X64A
[G6] 4072: ;Zeichen aus String lesen, Leerzeichen über
[PH] 4073: X64T: AND CX,CX
[0D] 4074: JE X64U
[CW] 4075: LODSB
[XG] 4076: CMP AL," "
[NR] 4077: LOOPE X64T
[3D] 4078: X64U: RET
[00] 4079:
[XF] 4080: ;Konvertierung Single -> Integer (zweitober
ster Wert, für logische Operationen)
[XF] 4081: L65: MOV BP,SP
[U3] 4082: MOV AX,SS:[BP+5]
[P3] 4083: MOV CH,SS:[BP+7]
[PI] 4084: AND CH,CH
[L9] 4085: JE X650
[ET] 4086: MOV BL,AH
[8H] 4087: OR AH,80H
[6R] 4088: MOV CL,144
[6S] 4089: SUB CL,CH
[D9] 4090: JC X650
[9P] 4091: SHR AX,CL
[YA] 4092: ADC AL,0
[Y9] 4093: AND BL,BL
[UG] 4094: JNS X650
[35] 4095: NEG AX
[W4] 4096: ;Oberste Integerzahl um die 2 freiwerdenden
Bytes nachrücken
[1S] 4097: X650: MOV SS:[BP+6],AX
[M3] 4098: MOV SS:[BP+2],AX
[P3] 4099: MOV SS:[BP+4],AX
[B2] 4100: RET 2
[00] 4101:
[3B] 4102: ;Prüft, ob übergebener String leer ist
[2E] 4103: L66: MOV BP,SP
[U4] 4104: ;Ergebnis initialisieren
[HN] 4105: XOR DX,DX
[FX] 4106: CMP BYTE PTR SS:[BP+2],0
[19] 4107: JE X650
[B4] 4108: DEC DX
[X4] 4109: X650: MOV SS:[BP+256],DX
[87] 4110: RET 254
[00] 4111:
[53] 4112: ;Funktion »MKIS«
[3K] 4113: L67: SUB SP,252
[2G] 4114: ;Platz für String schaffen
[8W] 4115: MOV BP,SP
[E3] 4116: ;Wert übertragen
[97] 4117: MOV AX,SS:[BP+254]
[E3] 4118: MOV SS:[BP+1],AX
[D7] 4119: ;Stringlänge ist 2
[LU] 4120: MOV BYTE PTR SS:[BP],2
[3N] 4121: JMP SS:[BP+252]
[00] 4122:
[16] 4123: ;Funktion »MKSS«
[H2] 4124: L68: SUB SP,250
```

```
[2G] 4125: ;Platz für String schaffen
[8W] 4126: MOV BP,SP
[12] 4127: ;Werte übertragen
[17] 4128: MOV AX,SS:[BP+252]
[E3] 4129: MOV SS:[BP+1],AX
[97] 4130: MOV AX,SS:[BP+254]
[W3] 4131: MOV SS:[BP+3],AX
[P4] 4132: ;Stringlänge schreiben
[N3] 4133: MOV BYTE PTR SS:[BP],4
[1N] 4134: JMP SS:[BP+250]
[00] 4135:
[7G] 4136: ;Funktion »RND«
[DG] 4137: L69: MOV BP,SP
[S1] 4138: MOV AX,SS:[BP+4]
[ND] 4139: ;Negatives Argument?
[RP] 4140: AND AL,AL
[ZH] 4141: JNS X690
[8C] 4142: ;Ja, dann Argument als neuen Startwert
[S1] 4143: MOV BX,SS:[BP+2]
[9H] 4144: MOV DS:[38],BX
[H2] 4145: MOV DS:[40],AX
[LD] 4146: ;Startwert auf Stack
[E2] 4147: X690: PUSH DS:[40]
[T2] 4148: PUSH DS:[38]
[HV] 4149: ;Argument 0, dann fertig
[SP] 4150: AND AH,AH
[AH] 4151: JE X695
[U4] 4152: ;Alte Zufallszahl mit Konstante multiplizie
ren
[53] 4153: MOV AX,98A7H
[F9] 4154: PUSH AX
[32] 4155: MOV AX,1B09H
[F9] 4156: PUSH AX
[LP] 4157: MOV DX,MULS-L00
[D6] 4158: CALL DX
[V8] 4159: ;Konstante addieren, um Nullergebnisse zu v
ermeiden
[F2] 4160: MOV AX,6828H
[F9] 4161: PUSH AX
[G6] 4162: MOV AX,0B146H
[F9] 4163: PUSH AX
[32] 4164: MOV DX,ADDS-L00
[D6] 4165: CALL DX
[13] 4166: ;Mantissenbytes manipulieren
[W3] 4167: MOV AX,SS:[BP+6]
[K2] 4168: MOV BX,SS:[BP+8]
[XK] 4169: XOR AL,0AAH
[CS] 4170: XOR BL,55H
[Y5] 4171: XCHG AL,BL
[8X] 4172: XOR AH,0C3H
[RE] 4173: MOV BH,0
[1L] 4174: AND AX,AX
[DP] 4175: JNE X691
[YF] 4176: AND BL,BL
[7A] 4177: JE X694
[FH] 4178: ;Ergebnis zwischen 0 und 1 bringen
[UN] 4179: X691: MOV BH,80H
[S3] 4180: ;Linksbindig machen
[F2] 4181: AND BL,BL
[NB] 4182: JS X693
[ND] 4183: SHL AX,1
[WC] 4184: RCL BL,1
[L3] 4185: DEC BH
[TG] 4186: JMP X692
[15] 4187: ;Vorzeichen ins Single-Format
[82] 4188: X692: AND BL,7FH
[N3] 4189: MOV SS:[BP+6],AX
[5P] 4190: MOV SS:[BP+8],BX
[P2] 4191: ;Ergebnis zurückschreiben und als neuen Sta
rtwert sichern
[XK] 4192: MOV BP,SP
[97] 4193: POP AX
[X3] 4194: MOV SS:[BP+6],AX
[5H] 4195: MOV DS:[38],AX
[97] 4196: POP AX
[33] 4197: MOV SS:[BP+8],AX
[1H] 4198: MOV DS:[40],AX
[3P] 4199: RET
[00] 4200:
[NW] 4201: ;Funktion »SCREEN«
[DH] 4202: L6A: MOV BP,SP
[97] 4203: ;Bildschirmseite lesen
[L7] 4204: MOV AH,0FH
[G8] 4205: INT 10H
[4W] 4206: ;Cursorposition holen
[7R] 4207: MOV AH,03H
[G8] 4208: INT 10H
[64] 4209: ;und sichern
[SX] 4210: MOV SI,DX
[2S] 4211: ;Cursor auf neue Position setzen
[S1] 4212: MOV DL,SS:[BP+4]
[S1] 4213: MOV DH,SS:[BP+6]
[B2] 4214: SUB DX,091H
[3R] 4215: MOV AH,02H
[G8] 4216: INT 10H
[S7] 4217: ;Zeichen und Attribut unter dem Cursor lese
n
[VR] 4218: MOV AH,08H
[G8] 4219: INT 10H
[97] 4220: ;Dritten Parameter abfragen (0 = Zeichen, 1
= Attribut)
[W3] 4221: MOV CX,SS:[BP+2]
[1T] 4222: AND CX,CX
[BAL] 4223: JE X6A0
[AT] 4224: MOV AL,AH
[P2] 4225: X6A0: MOV AH,0
[X4] 4226: ;Ergebnis zurückschreiben
[X3] 4227: MOV SS:[BP+6],AX
[0B] 4228: ;Cursorposition zurückholen
[WN] 4229: MOV DX,SI
[3R] 4230: MOV AH,02H
[G8] 4231: INT 10H
[L2] 4232: RET 4
[00] 4233:
[8C] 4234: ;Funktion »DATE$« (amerikanisches Format)
[R2] 4235: L6B: SUB SP,254
[2G] 4236: ;Platz für String schaffen
[8W] 4237: MOV BP,SP
[8X] 4238: ;MS-DOS-Funktion "Lies Datum"
[5T] 4239: MOV AH,2AH
[N8] 4240: INT 21H
[53] 4241: ;Monat
[L7] 4242: MOV AL,DH
[RG] 4243: CALL X6B1
[E3] 4244: MOV SS:[BP+1],AX
[E2] 4245: ;Trennzeichen
[S4] 4246: MOV BYTE PTR SS:[BP+3],"- "
[XN] 4247: ;Tag
[3U] 4248: MOV AL,DL
[RG] 4249: CALL X6B1
[R3] 4250: MOV SS:[BP+4],AX
[45] 4251: MOV BYTE PTR SS:[BP+5],"- "
[49] 4252: ;Jahr
[RS] 4253: MOV AX,"91"
[GT] 4254: SUB CX,1900
[84] 4255: ;Unterschneidung 19xx/29xx
[0N] 4256: CMP CX,100
[4A] 4257: JB X6B0
[F9] 4258: MOV AX,"02"
[0N] 4259: SUB CL,100
[R3] 4260: X6B0: MOV SS:[BP+7],AX
[0U] 4261: MOV AL,CL
[0U] 4262: CALL X6B1
[62] 4263: MOV SS:[BP+9],AX
[W2] 4264: ;Stringlänge
[RK] 4265: MOV BYTE PTR SS:[BP],10
[S2] 4266: JMP SS:[BP+254]
[K6] 4267: ;Unterprogramm "Byte in ASCII wandeln"
[V2] 4268: X6B1: MOV AH,0
[WN] 4269: MOV BL,10
[F6] 4270: DIV BL
```

```
[CL] 4271: ADD AX,"00"
[3P] 4272: RET
[00] 4273:
[PH] 4274: ;Funktion »TIMER«
[C8] 4275: L6C: SUB SP,2
[7B] 4276: ;BIOS-Interrupt "Lies Systemzeit"
[V9] 4277: MOV AH,00H
[1I] 4278: INT 1AH
[K1] 4279: ;Zurückgegebenen Wert ins Single-Format
[7U] 4280: MOV BL,DL
[2M] 4281: MOV DL,DH
[8U] 4282: MOV DH,CL
[ST] 4283: MOV CL,CH
[V1] 4284: MOV CH,0
[UG] 4285: ;Wert = 0?
[YF] 4286: AND BL,BL
[YF] 4287: JNE X6C0
[87] 4288: AND DX,DX
[YF] 4289: JNE X6C0
[L7] 4290: AND CX,CX
[SA] 4291: JE X6C3
[8M] 4292: X6C0: MOV CH,160
[N3] 4293: ;Linksbindig machen
[V2] 4294: X6C1: AND CL,CL
[J2] 4295: JS X6C2
[DD] 4296: SHL BL,1
[GD] 4297: RCL DX,1
[0D] 4298: RCL CL,1
[R3] 4299: DEC CH
[AH] 4300: PUSH AX
[RK] 4301: X6C2: AND CL,7FH
[S4] 4302: ;Wert mit 1/18.2 malnehmen
[L2] 4303: X6C3: PUSH CX
[V9] 4304: PUSH DX
[R3] 4305: MOV AX,7C60H
[F9] 4306: PUSH AX
[5C] 4307: MOV AX,0F97AH
[F9] 4308: PUSH AX
[LP] 4309: MOV DX,MULS-L00
[D6] 4310: CALL DX
[P4] 4311: ;Ergebnis zurückliefern
[F2] 4312: POP SS:[BP+10]
[97] 4313: POP AX
[6B] 4314: XCHG AX,SS:[BP+12]
[R6] 4315: JMP AX
[00] 4316:
[9H] 4317: ;Funktion »CVI«
[TH] 4318: L6D: MOV BP,SP
[U7] 4319: ;Ergebnis initialisieren
[WN] 4320: XOR AX,AX
[42] 4321: ;Stringlänge = 2
[W3] 4322: CMP BYTE PTR SS:[BP+2],2
[0G] 4323: JNE X6D0
[R2] 4324: ;Wert holen
[P3] 4325: MOV AX,SS:[BP+3]
[6D] 4326: ;und zurückliefern
[C4] 4327: X6D0: MOV SS:[BP+256],AX
[87] 4328: RET 254
[00] 4329:
[RK] 4330: ;Funktion »CVS«
[XH] 4331: L6E: MOV BP,SP
[U7] 4332: ;Ergebnis initialisieren
[WN] 4333: XOR AX,AX
[8U] 4334: MOV BX,0
[8B] 4335: ;Stringlänge < 4, dann 0 zurückgeben
[Y3] 4336: CMP BYTE PTR SS:[BP+2],4
[2G] 4337: JNE X6E0
[TH] 4338: ;Ergebnis aus String lesen
[P3] 4339: MOV AX,SS:[BP+3]
[Y1] 4340: MOV BX,SS:[BP+5]
[84] 4341: X6E0: MOV SS:[BP+254],AX
[F3] 4342: MOV SS:[BP+256],BX
[VY] 4343: ;String von Stack entfernen
[47] 4344: RET 252
[00] 4345:
[TW] 4346: ;Funktion »FIX« (Vorkommateilen zurückgebe
n)
[1K] 4347: L6F: MOV BP,SP
[DL] 4348: ;Sind Nachkommateilen vorhanden?
[B6] 4349: CMP BYTE PTR SS:[BP+5],152
[6F] 4350: JAE X6F3
[6B] 4351: ;Maske initialisieren
[D2] 4352: MOV AX,0FFFFH
[9Y] 4353: MOV BL,0FFH
[4A] 4354: ;Anzahl der Verschiebungen ermitteln
[6R] 4355: MOV CL,144
[6B] 4356: SUB CL,SS:[BP+5]
[L1] 4357: ;Maske verschieben
[XE] 4358: JAE X6F0
[P8] 4359: AND CL,8
[PS] 4360: MOV CL,CL
[FN] 4361: JMP SHORT X6F2
[U2] 4362: X6F0: MOV AL,0
[5E] 4363: SUB CL,8
[0F] 4364: JAE X6F1
[P8] 4365: AND CL,8
[BS] 4366: SHL AH,CL
[FN] 4367: JMP SHORT X6F2
[B9] 4368: X6F1: MOV AH,0
[39] 4369: CMP CL,8
[F9] 4370: JAE X6F4
[L1] 4371: SHL BL,CL
[GS] 4372: ;Wert mit Maske verknüpfen
[V8] 4373: X6F2: AND SS:[BP+2],AX
[XK] 4374: AND SS:[BP+4],BL
[9E] 4375: X6F3: RET
[46] 4376: ;Null zurückgeben
[U3] 4377: X6F4: MOV SS:[BP+2],AX
[R3] 4378: MOV SS:[BP+4],AX
[3P] 4379: RET
[00] 4380:
[53] 4381: ;Funktion »HEXS«
[AH] 4382: L70: SUB SP,252
[2G] 4383: ;Platz für String schaffen
[8W] 4384: MOV BP,SP
[XB] 4385: ;Integer-Wert holen
[R3] 4386: MOV AX,SS:[BP+254]
[H8] 4387: ;Wegen führender Nullen »DIV« und nicht »RO
L«
[ZP] 4388: MOV BX,16
[AH] 4389: XOR CX,CX
[D3] 4390: ;Die einzelnen Hexziffern bestimmen
[DK] 4391: X700: XOR DX,DX
[V6] 4392: DIV BX
[ZD] 4393: ;und auf Stack
[V9] 4394: PUSH DX
[LS] 4395: AND AX,AX
[KS] 4396: LOOPE X700
[B5] 4397: NEG CX
[S3] 4398: ;Stringlänge setzen
[HN] 4399: MOV SS:[BP],CL
[3P] 4400: MOV DI,1
[7F] 4401: ;Werte zurückholen
[V3] 4402: X701: POP AX
[77] 4403: ;Ins ASCII-Format wandeln
[6M] 4404: ADD AL,"0"
[81] 4405: CMP AL,"9"
[L7] 4406: JBE X702
[BB] 4407: AND AL,7
[F7] 4408: ;und in den String schreiben
[R4] 4409: X702: MOV SS:[BP+DI],AL
[15] 4410: INC DI
[VH] 4411: LOOP X701
[3N] 4412: JMP SS:[BP+252]
```

Listing 2. Der Quelltext der SBasic-Laufzeitbibliothek »sbasic.asm«


```
[000] 4413: ;Funktion *INP
[001] 4414: MOV BP,SP
[002] 4415: L71: MOV BP,SP
[003] 4416: ;Portadresse holen
[004] 4417: MOV DX,SS:[BP+2]
[005] 4418: ;Wert lesen
[006] 4419: IN AL,DX
[007] 4420: MOV AH,0
[008] 4421: ;Ergebnis zurückgeben
[009] 4422: MOV SS:[BP+2],AX
[010] 4423: RET
[011] 4424:
[012] 4425: ;WRITE= Zahlenstring
[013] 4426: L72: MOV BP,SP
[014] 4427: ADD BP,2
[015] 4428: ;Dateinummer
[016] 4429: MOV BX,SS:[BP+256]
[017] 4430: ;Stringlänge
[018] 4431: MOV CL,SS:[BP]
[019] 4432: MOV CH,0
[020] 4433: JCKX X722
[021] 4434: ;Ausgabe in Datei
[022] 4435: X720: INC BP
[023] 4436: MOV AL,SS:[BP]
[024] 4437: ;Leerzeichen überlesen
[025] 4438: CMP AL," "
[026] 4439: JE X721
[027] 4440: PUSH CX
[028] 4441: MOV DX,X006-L00
[029] 4442: CALL DX
[030] 4443: POP CX
[031] 4444: X721: LOOP X720
[032] 4445: ;String vom Stack
[033] 4446: X722: RET 256
[034] 4447:
[035] 4448: ;WRITE= String
[036] 4449: L73: MOV BP,SP
[037] 4450: ADD BP,2
[038] 4451: ;Dateinummer
[039] 4452: MOV BX,SS:[BP+256]
[040] 4453: ;Hochkomma ausgeben
[041] 4454: MOV AL,34
[042] 4455: MOV DX,X006-L00
[043] 4456: CALL DX
[044] 4457: ;Stringlänge
[045] 4458: MOV CL,SS:[BP]
[046] 4459: MOV CH,0
[047] 4460: JCKX X731
[048] 4461: ;Zeichen ausgeben
[049] 4462: X730: INC BP
[050] 4463: MOV AL,SS:[BP]
[051] 4464: PUSH CX
[052] 4465: MOV DX,X006-L00
[053] 4466: CALL DX
[054] 4467: POP CX
[055] 4468: LOOP X730
[056] 4469: ;String mit Hochkomma abschließen
[057] 4470: X731: MOV AL,34
[058] 4471: MOV DX,X006-L00
[059] 4472: CALL DX
[060] 4473: RET 256
[061] 4474:
[062] 4475: ;Ein Komma ausgeben
[063] 4476: L74: MOV BP,SP
[064] 4477: ;Dateinummer holen
[065] 4478: MOV BX,SS:[BP+2]
[066] 4479: AL: MOV DX,X006-L00
[067] 4480: JCKX X740
[068] 4481: JCKX X741
[069] 4482: ;WRITE=
[070] 4483: L75: MOV BP,SP
[071] 4484: ;Ersten Parameter holen
[072] 4485: MOV BL,SS:[BP+2]
[073] 4486: ;Zweiten Parameter holen
[074] 4487: MOV CL,SS:[BP+4]
[075] 4488: ;Portadresse
[076] 4489: MOV DX,SS:[BP+6]
[077] 4490: INC AL,DX
[078] 4491: XOR BL,BL
[079] 4492: AND AL,CL
[080] 4493: AND AL,CL
[081] 4494: JE X750
[082] 4495: RET 6
[083] 4496:
[084] 4497: ;Zuweisung an *TIMES=
[085] 4498: L76: MOV BP,SP
[086] 4499: ADD BP,3
[090] 4500: ;Stringlänge
[091] 4501: MOV CL,SS:[BP-1]
[092] 4502: MOV CH,0
[093] 4503: ;3 Zahlen lesen
[094] 4504: CALL X760
[095] 4505: MOV BL,BH
[096] 4506: CALL X760
[097] 4507: MOV BL,DX
[098] 4508: CALL X760
[099] 4509: MOV DL,0
[100] 4510: MOV CX,BX
[101] 4511: ;MS-DOS-Aufruf "Systemzeit setzen"
[102] 4512: MOV AH,2DH
[103] 4513: INT 21H
[104] 4514: RET 256
[105] 4515:
[106] 4516: ;Eine Zahl aus String lesen
[107] 4517: X760: MOV DH,0
[108] 4518: ;Auf gültige Dezimalziffer prüfen
[109] 4519: SUB AL,"0"
[110] 4520: JB X763
[111] 4521: CJP AL,10
[112] 4522: JAE X763
[113] 4523: ;Alten Wert mal 10
[114] 4524: PUSH AX
[115] 4525: MOV AL,10
[116] 4526: MUL DH
[117] 4527: MOV DH,AL
[118] 4528: POP AX
[119] 4529: ;Neue Ziffer dazuaddieren
[120] 4530: ADD DH,AL
[121] 4531: JCKX X761
[122] 4532: ;Ein Zeichen aus String lesen
[123] 4533: X762: MOV AL,0
[124] 4534: JCKX X763
[125] 4535: MOV AL,SS:[BP]
[126] 4536: INC BP
[127] 4537: DEC CX
[128] 4538: X763: RET
[129] 4539:
[130] 4540: ;SWAP= Integer
[131] 4541: L77: MOV BP,SP
[132] 4542: ;Adresse 1 holen
[133] 4543: MOV SI,SS:[BP+2]
[134] 4544: ;Adresse 2 holen
[135] 4545: MOV DI,SS:[BP+4]
[136] 4546: ;Zahlen vertauschen
[137] 4547: MOV AX,DS:[SI]
[138] 4548: XCHG AX,DS:[DI]
[139] 4549: MOV DS:[SI],AX
[140] 4550: ;Adressen vom Stack
[141] 4551: RET 4
[142] 4552:
[143] 4553: ;SWAP= Single
[144] 4554: L78: MOV BP,SP
[145] 4555: MOV AX,DS
[146] 4556: MOV ES,AX
[147] 4557: ;Variablenadressen lesen
[148] 4558: MOV SI,SS:[BP+2]
[149] 4559: MOV DI,SS:[BP+4]
[150] 4560: ;Zahlen vertauschen
[151] 4561: MOV AX,ES:[DI]
[152] 4562: MOVSW
[153] 4563: MOV DS:[SI-2],AX
```

```
[154] 4564: MOV AX,ES:[DI]
[155] 4565: MOVSW
[156] 4566: MOV DS:[SI-2],AX
[157] 4567: RET 4
[158] 4568:
[159] 4569: ;SWAP= String
[160] 4570: L79: MOV BP,SP
[161] 4571: MOV DL,DS
[162] 4572: ;Segment String 1
[163] 4573: MOV DS,SS:[BP+2]
[164] 4574: ;Segment String 2
[165] 4575: MOV ES,SS:[BP+4]
[166] 4576: ;Längen lesen
[167] 4577: MOV CL,DS:[0]
[168] 4578: MOV CH,ES:[0]
[169] 4579: ;Die längere Länge ist entscheidend
[170] 4580: CMP CL,CH
[171] 4581: JAE X790
[172] 4582: MOV CL,CH
[173] 4583: X790: MOV CH,0
[174] 4584: ;Längenbyte auch mitkopieren
[175] 4585: INC CX
[176] 4586: XOR SI,SI
[177] 4587: XOR DI,DI
[178] 4588: ;Strings vertauschen
[179] 4589: X791: MOV AL,ES:[DI]
[180] 4590: MOVSB
[181] 4591: MOV DS:[SI-1],AL
[182] 4592: LOOP X791
[183] 4593: MOV SI,DX
[184] 4594: RET 4
[185] 4595:
[186] 4596: ;SOUND=
[187] 4597: L7A: MOV BP,SP
[188] 4598: ;Dauer des Sounds berechnen
[189] 4599: MOV AX,9511H
[190] 4600: PUSH AX
[191] 4601: MOV AX,0A5F9H
[192] 4602: PUSH AX
[193] 4603: PUSH SS:[BP+8]
[194] 4604: PUSH SS:[BP+6]
[195] 4605: MOV DX,DIVS-L00
[196] 4606: CALL DX
[197] 4607: ;Ins Integer-Format wandeln
[198] 4608: POP AX
[199] 4609: POP BX
[200] 4610: AND BH,BH
[201] 4611: JE X7A1
[202] 4612: OR BL,00H
[203] 4613: MOV AL,AL
[204] 4614: MOV BH,BL
[205] 4615: X7A0: CMP BH,144
[206] 4616: JAE X7A1
[207] 4617: SHR AX,1
[208] 4618: INC BH
[209] 4619: JMP X7A0
[210] 4620: ;Frequenz ins Integer-Format
[211] 4621: X7A1: MOV BX,SS:[BP+14]
[212] 4622: AND BH,BH
[213] 4623: JE X7A3
[214] 4624: MOV CX,SS:[BP+12]
[215] 4625: OR BL,00H
[216] 4626: X7A2: CMP BH,144
[217] 4627: JAE X7A3
[218] 4628: SHR BL,1
[219] 4629: RCR CX,1
[220] 4630: INC BH
[221] 4631: JMP X7A2
[222] 4632: X7A3: AND BL,BL
[223] 4633: JNE X7A4
[224] 4634: JCKX X7A7
[225] 4635: X7A4: PUSH AX
[226] 4636: ;Timer 2 in Modus 3 schalten
[227] 4637: MOV AL,0B0H
[228] 4638: OUT 43H,AL
[229] 4639: ;Timer 0 in Modus 2 schalten
[230] 4640: MOV AL,14H
[231] 4641: OUT 43H,AL
[232] 4642: ;Timer 0 auf 256 Takte Zykluszeit umprogram-  
lieren
[233] 4643: MOV AL,0
[234] 4644: OUT 40H,AL
[235] 4645: ;Frequenz ausgeben
[236] 4646: POP AX
[237] 4647: OUT 42H,AL
[238] 4648: MOV AL,AL
[239] 4649: OUT 42H,AL
[240] 4650: ;Lautsprecher einschalten
[241] 4651: IN AL,61H
[242] 4652: OR AL,3
[243] 4653: OUT 61H,AL
[244] 4654: ;Einen Zyklus von Timer 0 abwarten
[245] 4655: X7A5: IN AL,40H
[246] 4656: AND AL,AL
[247] 4657: JNS X7A5
[248] 4658: X7A6: IN AL,40H
[249] 4659: AND AL,AL
[250] 4660: JS X7A6
[251] 4661: LOOP X7A5
[252] 4662: ;Tonende erreicht?
[253] 4663: DEC BL
[254] 4664: JNS X7A5
[255] 4665: ;Lautsprecher aus
[256] 4666: X7A7: IN AL,61H
[257] 4667: AND AL,0F0H
[258] 4668: OUT 61H,AL
[259] 4669: ;Timer 0 wieder auf Standardwerte
[260] 4670: MOV AL,36H
[261] 4671: OUT 43H,AL
[262] 4672: MOV AL,0
[263] 4673: OUT 40H,AL
[264] 4674: MOV AL,40H
[265] 4675: RET 8
[266] 4676:
[267] 4677: ;Einen Leerstring auf den Stack
[268] 4678: L7B: SUB SP,254
[269] 4679: ;Platz für String schaffen
[270] 4680: MOV BP,SP
[271] 4681: ;Stringlänge ist 0
[272] 4682: MOV BYTE PTR SS:[BP],0
[273] 4683: JMP SS:[BP+254]
[274] 4684:
[275] 4685: ;SHELL=
[276] 4686: L7C: MOV CX,DS:[15]
[277] 4687: ;Cursorform wiederherstellen
[278] 4688: MOV AH,01H
[279] 4689: INT 10H
[280] 4690: SUB SP,2
[281] 4691: MOV BP,SP
[282] 4692: ;Platz für "/C" schaffen
[283] 4693: MOV SI,SS:[BP+2]
[284] 4694: ;Stringlänge
[285] 4695: MOV AL,SS:[BP+4]
[286] 4696: MOV AH,13
[287] 4697: ;Mit "Carriage Return" terminierten Nullstr-  
ing schreiben
[288] 4698: MOV SS:[BP+1],AX
[289] 4699: MOV BYTE PTR SS:[BP+3],0
[290] 4700: ;Parameterstring leer?
[291] 4701: AND AL,AL
[292] 4702: JE X7C1
[293] 4703: ;Dem String ein "/C" voranstellen
[294] 4704: ADD AL,"/"
[295] 4705: MOV AH," "
[296] 4706: MOV SS:[BP+1],AX
[297] 4707: MOV WORD PTR SS:[BP+3],"/C"
[298] 4708: MOV AH,0
[299] 4709: MOV DI,AX
[300] 4710: CMB DI,256
[301] 4711: JBE X7C0
```

```
[302] 4712: MOV DI,256
[303] 4713: ;Kommandozeile mit "Carriage Return" abschli-  
eßen
[304] 4714: X7C0: MOV WORD PTR SS:[BP+DI+2],13
[305] 4715: X7C1: PUSH DS
[306] 4716: PUSH SI
[307] 4717: PUSH SS
[308] 4718: INC BP
[309] 4719: PUSH BP
[310] 4720: ;PSP-Adresse holen
[311] 4721: MOV DS,DS:[42]
[312] 4722: ;Environment-Segment
[313] 4723: MOV DS,DS:[2CH]
[314] 4724: PUSH DS
[315] 4725: ;Nach "COMSPEC" suchen
[316] 4726: XOR SI,SI
[317] 4727: CALL X7C2
[318] 4728: DB "COMSPEC",29
[319] 4729: X7C2: POP BX
[320] 4730: X7C3: MOV DX,SI
[321] 4731: LODSB
[322] 4732: ;Environment-Ende erreicht?
[323] 4733: AND AL,AL
[324] 4734: JE X7C7
[325] 4735: MOV DI,BX
[326] 4736: INC DI
[327] 4737: ;8 Zeichen vergleichen
[328] 4738: X7C4: AND AL,223
[329] 4739: INC DI
[330] 4740: CMOV AL,CS:[DI-1]
[331] 4741: LODSB
[332] 4742: LOOPX X7C4
[333] 4743: JE X7C6
[334] 4744: ;Nicht gefunden, bis zum Ende des momentane-  
n Eintrags lesen
[335] 4745: X7C5: LODSB
[336] 4746: AND AL,AL
[337] 4747: JNE X7C5
[338] 4748: ;Nächsten Eintrag überprüfen
[339] 4749: JMP X7C3
[340] 4750: ;Programmparameter für MS-DOS aufarbeiten
[341] 4751: X7C6: ADD DX,8
[342] 4752: X7C7: MOV AX,SS
[343] 4753: MOV ES,AX
[344] 4754: MOV SP,SP
[345] 4755: CALL X7C8
[346] 4756: POP DI
[347] 4757: ;Stack sichern
[348] 4758: MOV SS:[DI+X7C9+1-X7C8],SS
[349] 4759: MOV CS:[DI+X7CA+1-X7C8],SP
[350] 4760: ;MS-DOS-Funktion "Datei laden und ausführen"
[351] 4761: MOV AX,4B00H
[352] 4762: INT 21H
[353] 4763: ;Stack zurückholen
[354] 4764: CLI
[355] 4765: X7C9: MOV AX,0
[356] 4766: MOV SS,AX
[357] 4767: X7CA: MOV SP,0
[358] 4768: STI
[359] 4769: ;Segmente zurückholen
[360] 4770: MOV BP,SP
[361] 4771: MOV DS,SS:[BP+8]
[362] 4772: MOV BP,SS:[BP+6]
[363] 4773: ADD SP,270
[364] 4774: ;Bildschirmseite bestimmen
[365] 4775: MOV AH,0FH
[366] 4776: INT 10H
[367] 4777: ;Cursorposition lesen
[368] 4778: MOV AH,03H
[369] 4779: INT 10H
[370] 4780: ;Cursor in letzter Zeile, dann ein Zeichen  
nach oben
[371] 4781: CMP DH,24
[372] 4782: JB X7C7
[373] 4783: MOV AX,0E0AH
[374] 4784: INT 10H
[375] 4785: MOV DH,23
[376] 4786: MOV AH,02H
[377] 4787: INT 10H
[378] 4788: ;Cursorform setzen
[379] 4789: X7CB: MOV CX,DS:[36]
[380] 4790: MOV AH,01H
[381] 4791: INT 10H
[382] 4792: JMP BP
[383] 4793:
[384] 4794: ;SCREEN= (Grafik wird noch nicht unterstüt-  
zt)
[385] 4795: L7D: MOV BP,SP
[386] 4796: ;Bildschirmseite bestimmen
[387] 4797: MOV AH,0FH
[388] 4798: INT 10H
[389] 4799: ;Cursorposition lesen
[390] 4800: MOV AH,03H
[391] 4801: INT 10H
[392] 4802: ;Neue Bildschirmseite setzen
[393] 4803: MOV AL,SS:[BP+2]
[394] 4804: MOV AH,05H
[395] 4805: INT 10H
[396] 4806: ;Cursor auf alte Position setzen
[397] 4807: MOV AH,0FH
[398] 4808: INT 10H
[399] 4809: MOV AH,02H
[400] 4810: INT 10H
[401] 4811: RET 6
[402] 4812:
[403] 4813: ;CLEAR=
[404] 4814: L7E: MOV DX,DS
[405] 4815: ;Bereich für Systemvariablen (48 Byte) nich-  
t löschen
[406] 4816: ADD DX,3
[407] 4817: SUB BX,3
[408] 4818: JE X7E1
[409] 4819: ;Variablenbereich in Paragraphen-Schritten  
mit 0 füllen
[410] 4820: XOR AX,AX
[411] 4821: X7E0: MOV ES,DX
[412] 4822: XOR DI,DI
[413] 4823: MOV CX,0
[414] 4824: REP STOSW
[415] 4825: INC DX
[416] 4826: DEC BX
[417] 4827: JNE X7E0
[418] 4828: ;Alle offenen Dateien schließen
[419] 4829: X7E1: MOV BX,4
[420] 4830: MOV AX,X00X-L00
[421] 4831: CALL DX
[422] 4832: DEC BX
[423] 4833: JNE X7E2
[424] 4834: RET 4
[425] 4835:
[426] 4836: ;MKDIR=
[427] 4837: L7F: PUSH DS
[428] 4838: MOV AX,SS
[429] 4839: MOV AH,AX
[430] 4840: MOV DI,SP
[431] 4841: ADD DI,5
[432] 4842: ;Übergebenen String in das MS-DOS-Format (m-  
it 0 abschließen)
[433] 4843: MOV BL,DS:[DI-1]
[434] 4844: CMP BL,255
[435] 4845: JNE X7F0
[436] 4846: DEC BL
[437] 4847: MOV AH,BH
[438] 4848: MOV DS:[DI+BX],BH
[439] 4849: MOV DI,DI
[440] 4850: ;MS-DOS-Funktion aufrufen
[441] 4851: MOV AH,39H
[442] 4852: INT 21H
[443] 4853: POP DS
```



```
[C71] 4854: RET 256
[000] 4855:
[HV1] 4856: ;RMDIR*
[572] 4857: L80: PUSH DS
[SW1] 4858: MOV AX,SS
[XV1] 4859: MOV DS,AX
[9W1] 4860: MOV DI,SP
[EB1] 4861: ADD DI,5
[YB4] 4862: ;String ins MS-DOS-Format
[922] 4863: MOV BL,DS:[DI-1]
[SW1] 4864: CMP BL,255
[UE1] 4865: JNE X890
[R31] 4866: DEC BL
[572] 4867: X890: MOV BH,0
[R73] 4868: MOV DS:[DI+BX],BH
[FU1] 4869: MOV DX,DI
[176] 4870: ;MS-DOS-Funktion "Remove Directory"
[871] 4871: MOV AH,3AH
[NB1] 4872: INT 21H
[G71] 4873: POP DS
[C71] 4874: RET 256
[000] 4875:
[021] 4876: ;CHDIR*
[972] 4877: L81: PUSH DS
[SW1] 4878: MOV AX,SS
[XV1] 4879: MOV DS,AX
[9W1] 4880: MOV DI,SP
[EB1] 4881: ADD DI,5
[NU4] 4882: ;String mit 0 abschließen
[922] 4883: MOV BL,DS:[DI-1]
[SW1] 4884: CMP BL,255
[W11] 4885: JNE X810
[R31] 4886: DEC BL
[262] 4887: X810: MOV BH,0
[R73] 4888: MOV DS:[DI+BX],BH
[FU1] 4889: MOV DX,DI
[SG4] 4890: ;MS-DOS-Funktion aufrufen
[CT1] 4891: MOV AH,3BH
[NB1] 4892: INT 21H
[G71] 4893: POP DS
[C71] 4894: RET 256
[000] 4895:
[X22] 4896: ;RANDOMIZE*
[322] 4897: L82: POP DX
[FT3] 4898: ;Neuen Startwert lesen
[L12] 4899: POP DS:[38]
[F12] 4900: POP DS:[40]
[371] 4901: JMP DX
[000] 4902:
[292] 4903: ;BEEP*
[ST2] 4904: L83: MOV AX,0E97H
[AR4] 4905: ;Ton über das BIOS ausgeben
[GB1] 4906: INT 19H
[3P0] 4907: RET
[000] 4908:
[Y21] 4909: ;SPACES*
[XH2] 4910: L84: SUB SP,252
[2G4] 4911: ;Platz für String schaffen
[9W1] 4912: MOV DI,SP
[SW1] 4913: MOV ES,SS
[LW1] 4914: MOV ES,AX
[G15] 4915: ;Anzahl der Leerzeichen holen
[763] 4916: MOV AL,SS:[DI+254]
[L21] 4917: MOV BH,0
[YV1] 4918: MOV CX,AX
[R63] 4919: MOV DX,SS:[DI+252]
[HR3] 4920: ;Stringlänge sichern
[NW0] 4921: STOSB
[WU3] 4922: ;Leerzeichen schreiben
[FK1] 4923: MOV AL," "
[RK1] 4924: REP STOSB
[371] 4925: JMP DX
[000] 4926:
[471] 4927: ;BSAVE*
[3G2] 4928: L85: MOV BP,SP
[L22] 4929: ;Segment holen
[939] 4930: MOV ES,DS:[25]
[P91] 4931: PUSH DS
[SW1] 4932: MOV AX,SS
[XV1] 4933: MOV DS,AX
[DP2] 4934: LEA SI,[BP+6]
[GW3] 4935: ;Länge des Dateinamens
[HP2] 4936: MOV CL,DS:[SI]
[V11] 4937: MOV CH,0
[RN1] 4938: CMP CL,250
[C91] 4939: JB X850
[NW1] 4940: MOV CL,250
[W22] 4941: X850: ADD SI,CX
[Z22] 4942: LEA DI,[SI+1]
[TD1] 4943: CMP DI,0
[3E1] 4944: JBE X851
[BF1] 4945: MOV CL,4
[ME2] 4946: X851: JCXZ X853
[AG5] 4947: ;Ist eine Erweiterung vorhanden?
[MS4] 4948: X852: CMP BYTE PTR DS:[SI],""
[L11] 4949: JE X854
[P51] 4950: DEC SI
[9K1] 4951: LOOP X852
[545] 4952: ;Nein, dann *.BAS* anhängen
[YM4] 4953: X853: MOV WORD PTR DS:[DI],".B."
[RN4] 4954: MOV WORD PTR DS:[DI+2],".SA"
[L11] 4955: ADD SI,1
[N74] 4956: ;Namen mit 0 abschließen
[OP4] 4957: X854: MOV BYTE PTR DS:[DI],0
[Z22] 4958: LEA DX,[BP+7]
[DR1] 4959: MOV CX,20H
[A26] 4960: ;MS-DOS-Funktion "Legen neuer Datei an"
[G71] 4961: MOV AH,3CH
[NB1] 4962: INT 21H
[RK1] 4963: ;Fehler?
[L11] 4964: JC X856
[AF1] 4965: ;Parameter (Kennung FDhex, Segment, Offset und Länge) schreiben
[UW1] 4966: MOV BX,AX
[EV1] 4967: MOV AX,ES
[933] 4968: XCHG AX,SS:[BP+4]
[X13] 4969: MOV SS:[BP+6],AX
[Z11] 4970: MOV AH,0F0H
[N03] 4971: XCHG AX,SS:[BP+2]
[Z23] 4972: MOV SS:[BP+8],AX
[9C1] 4973: MOV CX,7
[RD2] 4974: LEA DX,[BP+3]
[R71] 4975: MOV AH,40H
[NB1] 4976: INT 21H
[X55] 4977: ;Schreibfehler oder Diskette voll?
[Y91] 4978: JC X855
[AU1] 4979: CMP AX,CX
[L11] 4980: JNE X855
[4K3] 4981: ;Datenblock schreiben
[Z13] 4982: MOV DS,SS:[BP+4]
[823] 4983: MOV DX,SS:[BP+6]
[Z23] 4984: MOV CX,SS:[BP+8]
[R71] 4985: MOV AH,40H
[NB1] 4986: INT 21H
[S13] 4987: ;Datei schließen
[8R2] 4988: X855: MOV AH,3EH
[NB1] 4989: INT 21H
[C42] 4990: X856: POP DS
[L71] 4991: RET 260
[000] 4992:
[FR1] 4993: ;BLOAD*
[7G2] 4994: L86: MOV BP,SP
[B83] 4995: ;Basic-Datensegment
[DH2] 4996: MOV ES,DS:[25]
[P91] 4997: PUSH DS
[SW1] 4998: MOV AX,SS
[XV1] 4999: MOV DS,AX
[W22] 5000: ;Stringlänge
[HP2] 5001: MOV CL,DS:[SI]
```

```
[VE1] 5003: MOV CH,0
[RN1] 5004: CMP CL,250
[E91] 5005: JB X860
[9R1] 5006: MOV CL,250
[0G2] 5007: X860: ADD SI,CX
[Z22] 5008: LEA DI,[SI+1]
[TD1] 5009: CMP CL,4
[5E1] 5010: JBE X861
[BF1] 5011: MOV CL,4
[TE2] 5012: X861: JCXZ X863
[AG5] 5013: ;Ist eine Erweiterung vorhanden?
[5S4] 5014: X862: CMP BYTE PTR DS:[SI],""
[3A1] 5015: JE X864
[5S1] 5016: DEC SI
[BK1] 5017: LOOP X862
[P45] 5018: ;Nein, dann *.BAS* anhängen
[ZK4] 5019: X863: MOV WORD PTR DS:[DI],".B."
[RN4] 5020: MOV WORD PTR DS:[DI+2],".SA"
[BK1] 5021: ADD DI,4
[YB4] 5022: ;String ins MS-DOS-Format
[4P4] 5023: X864: MOV BYTE PTR DS:[DI],0
[XD2] 5024: LEA DX,[BP+5]
[714] 5025: ;Datei zum Lesen öffnen
[432] 5026: MOV AX,3D00H
[NB1] 5027: INT 21H
[Y23] 5028: ;Fehler: Nicht gefunden
[9A1] 5029: JC X868
[UV1] 5030: MOV BX,AX
[PY2] 5031: ;Datei-Daten lesen
[0G1] 5032: MOV CX,7
[WT1] 5033: MOV AH,3FH
[NB1] 5034: INT 21H
[6A1] 5035: JC X867
[6E1] 5036: CMP AX,7
[VF1] 5037: JNE X867
[SP5] 5038: ;Ist die Beave-Kennung vorhanden?
[ED4] 5039: CMP BYTE PTR DS:[BP+5],0F0H
[VE1] 5040: JNE X867
[ZA5] 5041: ;Ist eine Ladeadresse angegeben?
[023] 5042: MOV DX,SS:[BP+2]
[SL1] 5043: CMP DX,-1
[NP1] 5044: JNE X865
[LVS] 5045: ;Ja, dann Dateiparameter übergehen
[Z23] 5046: MOV DS,SS:[BP+6]
[C23] 5047: MOV DX,SS:[BP+8]
[TL2] 5048: JMP SHORT X866
[SK2] 5049: X865: MOV AX,ES
[XV1] 5050: MOV DS,AX
[Y22] 5051: ;Anzahl der Bytes
[6W3] 5052: X866: MOV CX,SS:[BP+10]
[LY1] 5053: ;Block lesen
[WT1] 5054: MOV AH,3FH
[NB1] 5055: INT 21H
[S13] 5056: ;Datei abschließen
[CR2] 5057: X867: MOV AH,3EH
[NB1] 5058: INT 21H
[K42] 5059: X868: POP DS
[CT1] 5060: RET 258
[000] 5061:
[7X7] 5062: ;Randfarbe setzen (3. Parameter von *COLOR*
[BG2] 5063: L87: MOV BP,SP
[R21] 5064: ;Wert holen
[C13] 5065: MOV BL,SS:[BP+2]
[R21] 5066: MOV BH,0
[TG6] 5067: ;Entsprechende BIOS-Funktion aufrufen
[371] 5068: MOV AL,BBH
[G81] 5069: INT 09H
[B21] 5070: RET 2
[000] 5071:
[6P1] 5072: ;OUT*
[FC2] 5073: L88: MOV BP,SP
[ED2] 5074: ;Adresse lesen
[Z23] 5075: MOV DX,SS:[BP+4]
[B02] 5076: ;Wert lesen
[R13] 5077: MOV AL,SS:[BP+2]
[W12] 5078: ;und ausgeben
[OV1] 5079: OUT DX,AL
[L21] 5080: RET 4
[000] 5081:
[M22] 5082: ;PLAY*
[GN2] 5083: L89: MOV AL,0B0H
[Z23] 5084: ;Timer 2 in Modus 3
[MS1] 5085: OUT 43H,AL
[023] 5086: ;Timer 9 in Modus 2
[KR1] 5087: MOV AL,14H
[MS1] 5088: OUT 43H,AL
[L85] 5089: ;Timer 1 auf 256 Zyklen schalten
[R21] 5090: MOV AL,0
[MS1] 5091: OUT 40H,AL
[8W1] 5092: MOV BP,SP
[W2A] 5093: ;Stringlänge
[G13] 5094: MOV CL,SS:[BP+2]
[7B1] 5095: ADD BP,3
[R11] 5096: CALL X89X
[4R3] 5097: ;Stringende erreicht?
[E22] 5098: X899: AND AL,AL
[P71] 5099: JNE X891
[Z17] 5100: ;Ja, dann Timer 0 auf Standardwerte setzen
[0S1] 5101: MOV AL,36H
[MS1] 5102: OUT 43H,AL
[R21] 5103: MOV AL,0
[MS1] 5104: OUT 40H,AL
[MS1] 5105: OUT 40H,AL
[UV4] 5106: ;String vom Stack entfernen
[CT1] 5107: RET 256
[C94] 5108: ;Ist es eine Note A bis G?
[WG2] 5109: X891: CMP AL,"A"
[3A1] 5110: JB X895
[TN1] 5111: CMP AL,"G"
[0A1] 5112: JA X895
[L11] 5113: SUB AL,"A"
[L21] 5114: MOV AH,0
[XW8] 5115: ;Ja, dann Buchstabe in Oktaven-Offset 0 bis 11 umrechnen
[5G1] 5116: CALL X892
[L22] 5117: DB 10,12,1,3,5,6,8
[652] 5118: X892: POP BX
[R21] 5119: ADD BX,AX
[FP2] 5120: MOV AL,CS:[BX]
[XUB] 5121: ;Eingestellte Oktave (Vielfaches von 12) da zuaddieren
[R22] 5122: ADD AL,DS:[44]
[SG2] 5123: X893: MOV BX,AL
[L11] 5124: CALL X89X
[V66] 5125: ;Ist es ein Halbton-Erhöungszeichen?
[ZF3] 5126: X894: MOV BH,DS:[45]
[SH1] 5127: CMP AL,"#"
[4G1] 5128: JNE X898
[Z35] 5129: ;Note um einen Halbton erhöhen
[W41] 5130: INC BL
[UK1] 5131: JCMP X899
[8A3] 5132: ;Ist es eine Pause?
[DK2] 5133: X895: CMP AL,"P"
[YF1] 5134: JNE X896
[CD4] 5135: ;Note 0 spielen (C0 = Pause)
[R21] 5136: MOV AL,0
[YG1] 5137: JMP X893
[SP5] 5138: ;Soll die Oktave gesetzt werden?
[BK2] 5139: X896: CMP AL,"O"
[L11] 5140: JNE X897
[L11] 5141: CALL X89X
[K54] 5142: ;Ja, dann Parameter lesen
[3G1] 5143: CALL X890
[F91] 5144: PUSH AX
[Z24] 5145: ;Oktave mit 12 multiplizieren
[MS1] 5146: MOV AL,12
[W71] 5147: MUL DL
[642] 5148: ;und sichern
[7G2] 5149: MOV DS:[44],AL
```

```
[971] 5150: POP AX
[N61] 5151: JMP X890
[0P5] 5152: ;Ist es eine Notendirektangabe?
[9K2] 5153: X897: CMP AL,"N"
[7G1] 5154: JNE X899
[R11] 5155: CALL X89X
[R63] 5156: ;Notennummer lesen
[3G1] 5157: CALL X890
[7U1] 5158: MOV BL,DL
[L11] 5159: JMP X890
[P05] 5160: ;Soll die Note erhöht werden?
[P22] 5161: X898: CMP AL,"+"
[EH1] 5162: JNE X89F
[W41] 5163: INC BL
[PW2] 5164: JMP SHORT X89G
[CP4] 5165: ;Ist es eine Längenangabe?
[5K2] 5166: X899: CMP AL,"L"
[ZG1] 5167: JNE X89A
[R11] 5168: CALL X89X
[0P5] 5169: ;Ja, dann Wert lesen und sichern
[3G1] 5170: CALL X890
[6H2] 5171: MOV DS:[45],DL
[N61] 5172: JMP X890
[B84] 5173: ;Ist es eine Tempo-Angabe?
[5L2] 5174: X89A: CMP AL,"T"
[ZH1] 5175: JNE X89B
[R11] 5176: CALL X89X
[L34] 5177: ;Tempo lesen und sichern
[3G1] 5178: CALL X890
[RH2] 5179: MOV DS:[46],DL
[N61] 5180: JMP X890
[G66] 5181: ;Soll die Oktave verringert werden?
[YG2] 5182: X89B: CMP AL,"<"
[8H1] 5183: JNE X89D
[S36] 5184: ;Darf die Oktave verringert werden?
[P73] 5185: CMP BYTE PTR DS:[44],12
[L11] 5186: JB X89C
[V14] 5187: ;Oktave um 1 verringern
[FT3] 5188: SUB BYTE PTR DS:[44],12
[Z22] 5189: X89C: CALL X890
[N61] 5190: JMP X890
[6L3] 5191: ;Oktave um 1 erhöhen?
[6H2] 5192: X89D: CMP AL,">"
[NH1] 5193: JNE X89E
[N73] 5194: CMP BYTE PTR DS:[44],72
[4G1] 5195: JAE X89E
[0R3] 5196: ADD BYTE PTR DS:[44],12
[Z22] 5197: X89E: CALL X89X
[N61] 5198: JMP X890
[8B8] 5199: ;Soll die Note um einen Halbton verringert werden?
[NF2] 5200: X89F: CMP AL,"-"
[MH1] 5201: JNE X89H
[R11] 5202: DEC BL
[L11] 5203: CALL X89X
[Z25] 5204: ;Folgt ein numerischer Parameter?
[0G2] 5205: X89H: CMP AL,"0"
[ZB1] 5206: JB X891
[LMI] 5207: CMP AL,"0"
[W11] 5208: JA X891
[6E4] 5209: ;Ja, dann Notenlänge lesen
[3G1] 5210: CALL X890
[ZU1] 5211: MOV BL,DL
[NB2] 5212: X891: MOV AH,0
[8B5] 5213: ;Soll die Note gestreckt werden?
[W22] 5214: X89J: CMP AL,"."
[9K1] 5215: JNE X890
[G45] 5216: ;Anzahl der Punkte um 1 erhöhen
[M41] 5217: INC AH
[R11] 5218: CALL X89X
[Z11] 5219: JMP X89J
[M64] 5220: ;Ist es ein Modus-Befehl?
[UK2] 5221: X89K: CMP AL,"M"
[6K1] 5222: JNE X89N
[R11] 5223: CALL X89X
[XR2] 5224: ;Normalisieren?
[NP1] 5225: CMP AL,"N"
[0K1] 5226: JNE X89L
[ZN2] 5227: ;Notenlänge 7/8
[PS3] 5228: MOV BYTE PTR DS:[47],7
[XM2] 5229: JMP SHORT X89N
[6B3] 5230: ;Legato (gebunden)?
[SK2] 5231: X89L: CMP AL,"L"
[3K1] 5232: JNE X89N
[VW3] 5233: ;Noten voll ausspielen
[AS3] 5234: MOV BYTE PTR DS:[47],8
[XM2] 5235: JMP SHORT X89N
[TE3] 5236: ;Staccato (zerhackt)?
[EL2] 5237: X89M: CMP AL,"S"
[6K1] 5238: JNE X89N
[UG3] 5239: ;Eine Note ist 6/8 lang
[8S3] 5240: MOV BYTE PTR DS:[47],6
[N61] 5241: X89N: CALL X89X
[NG1] 5242: JMP X890
[TE2] 5243: X89O: CALL X89P
[P39] 5244: ;Tabelle der einzelnen Frequenzen in Halbtonschritten
[977] 5245: DW 1,18243,17219,16252,15340,1447
9,13666,12899,12175,11492
[LMT] 5246: DW 10847,10238,9664,9121,8609,812
6,7670,7240,6833,6450,6088
[ZW7] 5247: DW 5746,5424,5119,4832,4561,4305,
4063,3835,3620,3417,3225,3044
[MV7] 5248: DW 2873,2712,2560,2416,2280,2152,
2032,1918,1810,1708,1612,1522
[LV7] 5249: DW 1437,1356,1280,1208,1140,1076,
1016,959,905,854,806,761,718
[MV7] 5250: DW 678,640,604,570,538,508,479,45
2,427,403,380,359,339,320,302
[H86] 5251: DW 285,269,254,240,226,214,202,19
0,189,169,160,151
[V52] 5252: X89P: POP SI
[MF4] 5253: ;Index auf Tabelle berechnen
[F91] 5254: PUSH AX
[X71] 5255: MOV AL,BL
[L21] 5256: MOV AH,0
[ND1] 5257: SHL AX,1
[ET1] 5258: ADD SI,AX
[UR2] 5259: ;Frequenz holen
[R22] 5260: MOV AX,CS:[SI]
[L72] 5261: ;und ausgeben
[LS1] 5262: OUT 42H,AL
[AT1] 5263: MOV AL,AH
[LS1] 5264: OUT 42H,AL
[HT1] 5265: MOV BL,BH
[DR3] 5266: ;Notendauer berechnen
[R21] 5267: MOV BH,0
[TD1] 5268: SHL BX,1
[TD1] 5269: SHL BX,1
[TD1] 5270: SHL BX,1
[TD1] 5271: SHL BX,1
[971] 5272: POP AX
[P91] 5273: PUSH AX
[AT1] 5274: MOV AL,AH
[LE1] 5275: MOV AH,0
[LVI] 5276: MOV DI,AX
[2R1] 5277: MOV BL,160
[CG3] 5278: MUL BYTE PTR DS:[47]
[WV1] 5279: MOV SI,DI
[X32] 5280: X89Q: DEC SI
[DE1] 5281: JS X89R
[SF1] 5282: MOV DX,3
[H71] 5283: MUL DX
[ET1] 5284: SHR DX,1
[WD1] 5285: RCR AX,1
```

Listing 2. Der Quelltext der SBasic-Laufzeitbibliothek »sbasic.asm«


```
[SL1] 5286: JMP X89Q
[SG3] 5287: X89R: MOV DL,DS:[46]
[2B1] 5288: MOV DH,0
[HT1] 5289: MUL DX
[VG1] 5290: DIV BX
[FX1] 5291: MOV SI,AX
[AL1] 5292: MOV AX,160*8
[AG2] 5293: X89S: DEC DI
[LE1] 5294: JS X89T
[SP1] 5295: MOV DX,3
[HT1] 5296: MUL DX
[TE1] 5297: SHR DX,1
[WD1] 5298: RCR AX,1
[YL1] 5299: JNC X89S
[VG3] 5300: X89T: MOV DL,DS:[46]
[2B1] 5301: MOV DH,0
[HT1] 5302: MUL DX
[VE1] 5303: DIV BX
[LV1] 5304: MOV DI,AX
[2B4] 5305: ;Lautesprecher einschalten
[KG1] 5306: IN AL,61H
[NB1] 5307: OR AL,3
[US1] 5308: OUT 61H,AL
[RNS] 5309: ;Einen Zyklus von Timer 0 abwarten
[IK2] 5310: X89U: IN AL,40H
[RP1] 5311: AND AL,AL
[2B1] 5312: JS X89U
[2K2] 5313: X89V: IN AL,40H
[RP1] 5314: AND AL,AL
[PM1] 5315: JNS X89V
[TK5] 5316: ;Zykluszähler "Ton an" verringern
[SS1] 5317: DEC SI
[LL1] 5318: JNE X89V
[SC4] 5319: ;Lautesprecher ausschalten
[KG1] 5320: IN AL,61H
[IM1] 5321: AND AL,253
[US1] 5322: OUT 61H,AL
[R07] 5323: ;Zykluszähler "Gesamtlänge" verringern
[222] 5324: X89W: DEC DI
[VK1] 5325: JNE X89U
[971] 5326: POP AX
[NG1] 5327: JNC X89Q
[OK6] 5328: ;Ein Zeichen aus Parameterstring lesen
[N92] 5329: X89X: MOV AL,0
[FP1] 5330: JCXZ X89Z
[2P2] 5331: MOV AL,SS:[BP]
[XK8] 5332: ;Zeichen möglicherweise in Großbuchstaben wandeln
[IN1] 5333: CMP AL,"a"
[FD1] 5334: JB X89Y
[5S1] 5335: CMP AL,"z"
[CD1] 5336: JB X89Y
[UU1] 5337: AND AL,0DFH
[X32] 5338: X89Y: INC BP
[741] 5339: DEC CX
[2E1] 5340: X89Z: RET
[EM4] 5341: ;Eine numerische Angabe lesen
[AW2] 5342: X89Q: MOV DX,0A00H
[IF2] 5343: X89I: SUB AL,"0"
[471] 5344: XCHG AL,DL
[PK5] 5345: ;Alten Wert mit 10 multiplizieren
[171] 5346: MUL DH
[N44] 5347: ;Neuen Wert dazuaddieren
[SP1] 5348: ADD DL,AL
[7S5] 5349: ;Ist nächstes Zeichen eine Ziffer?
[RL1] 5350: CALL X89Q
[XK1] 5351: CMP AL,"0"
[Y91] 5352: JB X89Z
[IM1] 5353: CMP AL,"9"
[FE1] 5354: JBE X89I
[WCL] 5355: X89Z: RET
[000] 5356:
[005] 5357: ;Funktion *ABS* (Integer)
[LH2] 5358: L8B: MOV BP,SP
[SC2] 5359: ;Zahl negativ
[494] 5360: TEST BYTE PTR SS:[BP+3],80H
[MH1] 5361: JNS X8A0
[YW2] 5362: ;Ja, dann negieren
[BR3] 5363: NEG WORD PTR SS:[BP+2]
[ED1] 5364: X8A0: RET
[000] 5365:
[7X4] 5366: ;Funktion *ABS* (Single)
[RH2] 5367: L8B: MOV BP,SP
[P94] 5368: ;Vorzeichenbit löschen
[2B4] 5369: AND BYTE PTR SS:[BP+4],7FH
[3P9] 5370: RET
[000] 5371:
[C12] 5372: ;XILL:
[RH2] 5373: L8C: PUSH DS
[SW1] 5374: MOV AX,SS
[XV1] 5375: MOV DS,AX
[9W1] 5376: MOV DI,SP
[EB1] 5377: ADD DI,5
[W22] 5378: ;Stringlänge
[922] 5379: MOV BL,DS:[DI-1]
[SN1] 5380: CMP BL,255
[OG1] 5381: JNE X8C0
[R31] 5382: DEC AL
[5E5] 5383: ;Dateinamen mit 0 abschließen
[792] 5384: X8C0: MOV BH,0
[R73] 5385: MOV DS:[DI+BX],BH
[FU1] 5386: MOV DX,DI
[L85] 5387: ;MS-DOS-Funktion "Lösche Datei"
[BR1] 5388: MOV AH,41H
[NB1] 5389: INT 21H
[G71] 5390: POP DS
[C71] 5391: RET 256
[000] 5392:
[SU1] 5393: ;FILES:
[2H2] 5394: L8D: MOV BP,SP
[7B1] 5395: ADD BP,3
[1W2] 5396: ;Stringlänge = 0?
[C13] 5397: MOV BL,SS:[BP-1]
[YP1] 5398: AND BL,BL
[2G1] 5399: JNE X8D0
[2D5] 5400: ;Ja, dann ... als Suchmaske
[745] 5401: MOV WORD PTR SS:[BP],..."
[844] 5402: MOV BYTE PTR SS:[BP+2],..."
[4P1] 5403: MOV BL,3
[R02] 5404: X8D0: CMP BL,255
[5G1] 5405: JNE X8D1
[R31] 5406: DEC BL
[D56] 5407: ;String mit Nullbyte abschließen
[C92] 5408: X8D1: MOV BH,0
[W11] 5409: ADD BX,BP
[3P1] 5410: MOV DL,0
[XN2] 5411: MOV SS:[BX],DL
[076] 5412: ;Ist eine Laufwerkangabe vorhanden?
[4N3] 5413: X8D2: MOV AL,SS:[BP]
[0P1] 5414: INC BP
[RP1] 5415: AND AL,AL
[W11] 5416: JE X8D3
[SW1] 5417: CMP AL,"*"
[EG1] 5418: JNE X8D4
[SP5] 5419: ;Ja, dann line Format 1 bis n bringen
[UU1] 5420: AND AL,0DFH
[XM1] 5421: MOV AL,"*"
[LA1] 5422: JB X8D3
[3M1] 5423: CMP AL,26
[G11] 5424: JA X8D3
[6U1] 5425: MOV DI,AL
[FN2] 5426: JMP SHORT X8D4
[4A2] 5427: X8D3: MOV DL,0
[PM4] 5428: ;Ist ein Laufwerk angegeben?
[OG2] 5429: X8D4: AND DL,DL
[HC1] 5430: JNE X8D5
[9U5] 5431: ;Nein, dann Standardlaufwerk holen
[2S1] 5432: MOV AH,19H
[NB1] 5433: INT 21H
[6U1] 5434: MOV DL,AL
```

```
[451] 5435: INC DL
[GC2] 5436: X8D5: PUSH DX
[A74] 5437: ;Dateiname 0 (*SCRN*)
[3W1] 5438: XOR BX,BX
[MB6] 5439: ;Laufwerksbuchstaben in ASCII wandeln
[3U1] 5440: MOV AL,DL
[EK1] 5441: ADD AL,"e"
[172] 5442: ;und ausgeben
[AG2] 5443: MOV DX,X006-L00
[D61] 5444: CALL DX,"*"
[RN1] 5445: MOV AL,"*"
[AG2] 5446: MOV DX,X006-L00
[D61] 5447: CALL DX
[271] 5448: MOV AL,"*"
[AG2] 5449: MOV DX,X006-L00
[D61] 5450: CALL DX
[N71] 5451: POP DX
[V91] 5452: PUSH DX
[4B7] 5453: ;Platz für momentanes Verzeichnis schaffen
[LP1] 5454: SUB SP,64
[Y11] 5455: MOV SI,SP
[P91] 5456: PUSH DS
[SW1] 5457: MOV AX,SS
[XV1] 5458: MOV DS,AX
[S53] 5459: ;Verzeichnis lesen
[3S1] 5460: MOV AH,47H
[NB1] 5461: INT 21H
[G71] 5462: POP DS
[N93] 5463: ;Verzeichnis ausgeben
[NN3] 5464: X8D6: MOV AL,SS:[SI]
[X61] 5465: INC SI
[RP1] 5466: AND AL,AL
[8B1] 5467: JE X8D7
[AG2] 5468: MOV DX,X006-L00
[D61] 5469: CALL DX
[XH1] 5470: JMP X8D6
[NB2] 5471: X8D7: ADD SP,64
[A33] 5472: ;In neue Zeile gehen
[6N1] 5473: MOV AL,13
[AG2] 5474: MOV DX,X006-L00
[D61] 5475: CALL DX
[B29] 5476: ;MS-DOS-Funktion "Hole DTA-Adresse" (*Disk Transfer Area*)
[TT1] 5477: MOV AH,2FH
[NB1] 5478: INT 21H
[BM2] 5479: LEA SI,[BX+1EH]
[3W1] 5480: XOR BX,BX
[P91] 5481: PUSH DS
[SW1] 5482: MOV AX,SS
[XV1] 5483: MOV DS,AX
[SW1] 5484: MOV DX,SP
[3C1] 5485: ADD DX,7
[ES6] 5486: ;MS-DOS-Funktion "Suche ersten Eintrag"
[NR1] 5487: MOV CL,31H
[VT1] 5488: MOV AH,4EH
[NB1] 5489: INT 21H
[G71] 5490: POP DS
[X94] 5491: ;Noch ein Name vorhanden?
[192] 5492: X8D8: JNC X8D9
[LL1] 5493: JMP X8D9H
[7L2] 5494: X8D9: XOR BP,BP
[DV1] 5495: MOV DI,SI
[ACA] 5496: ;Nächstes Zeichen lesen
[WX3] 5497: X8DA: MOV ES:[BP+DI]
[RP1] 5498: AND AL,AL
[FC1] 5499: JE X8DD
[G53] 5500: ;Ist es ein Punkt?
[SW1] 5501: CMP AL,"."
[CC1] 5502: JE X8DC
[B93] 5503: ;Nein, dann ausgeben
[FC3] 5504: X8DB: MOV DX,X006-L00
[D61] 5505: CALL DX
[9S1] 5506: INC BP
[YK1] 5507: JMP X8DA
[GNA] 5508: ;Eventuell Namen bis zur Position 8 mit Leezeichen erfüllen
[1K2] 5509: X8DC: MOV AL,"."
[SE1] 5510: CMP BP,8
[NG1] 5511: JAE X8DB
[PK1] 5512: MOV AL,"."
[AG2] 5513: MOV DX,X006-L00
[D61] 5514: CALL DX
[9S1] 5515: INC BP
[W31] 5516: DEC DI
[AND] 5517: JNC X8DC
[GR3] 5518: ;Erweiterung ausgeben
[RE2] 5519: X8DD: CMP BP,12
[UG1] 5520: JAE X8DE
[FR1] 5521: MOV AL,"."
[AG2] 5522: MOV DX,X006-L00
[D61] 5523: CALL DX
[9S1] 5524: INC BP
[7L1] 5525: JMP X8DD
[CB2] 5526: X8DE: MOV BP,6
[R55] 5527: ;Ist es ein Unterverzeichnis?
[294] 5528: TEST BYTE PTR ES:[SI-9],10H
[NL1] 5529: JNE X8DF
[TS4] 5530: ;Ja, dann "DIR" ausgeben
[ZN1] 5531: MOV AL,"*"
[AG2] 5532: MOV DX,X006-L00
[D61] 5533: CALL DX
[ZP1] 5534: MOV AL,"d"
[AG2] 5535: MOV DX,X006-L00
[D61] 5536: CALL DX
[LR1] 5537: MOV AL,"i"
[AG2] 5538: MOV DX,X006-L00
[D61] 5539: CALL DX
[RS1] 5540: MOV AL,"r"
[AG2] 5541: MOV DX,X006-L00
[D61] 5542: CALL DX
[7P1] 5543: MOV AL,"*"
[AG2] 5544: MOV DX,X006-L00
[D61] 5545: CALL DX
[ZP1] 5546: MOV BP,1
[DH5] 5547: ;Nein, dann 6 Leerzeichen ausgeben
[U22] 5548: X8DF: MOV AL," "
[AG2] 5549: MOV DX,X006-L00
[D61] 5550: CALL DX
[V31] 5551: DEC BP
[4K1] 5552: JNE X8DF
[YW3] 5553: ;Bildschirmseite holen
[LT1] 5554: MOV AH,0FH
[XH3] 5555: INT 10H
[GR1] 5556: ;Cursorposition holen
[TR1] 5557: MOV AH,03H
[XH3] 5558: INT 10H
[R55] 5559: ;Paßt noch ein Name in die Zeile?
[3M1] 5560: XOR BX,BX
[SK1] 5561: ADD BX,1
[XE2] 5562: CMP DX,[SI+1]
[YH1] 5563: JB X8DG
[125] 5564: ;Nein, dann in neue Zeile gehen
[6N1] 5565: MOV AL,13
[AG2] 5566: MOV DX,X006-L00
[D61] 5567: CALL DX
[3C4] 5568: ;Nächsten Eintrag lesen
[TT2] 5569: X8DG: MOV AH,4FH
[NB1] 5570: INT 21H
[3K1] 5571: JMP X8DB
[YW3] 5572: ;Bildschirmseite holen
[K72] 5573: X8DH: MOV AH,0FH
[G81] 5574: INT 10H
[AW3] 5575: ;Cursorposition lesen
[7R1] 5576: MOV AH,03H
[XH3] 5577: INT 10H
[3W1] 5578: XOR BX,BX
[9A6] 5579: ;Steht der Cursor am Anfang der Zeile?
[CR1] 5580: AND DL,DL
[YC1] 5581: JE X8DI
[YG5] 5582: ;Nein, dann in nächste Zeile gehen
```

```
[6N1] 5583: MOV AL,13
[AG2] 5584: MOV DX,X006-L00
[D61] 5585: CALL DX
[XG2] 5586: X8DI: MOV AL," "
[AG2] 5587: MOV DX,X006-L00
[D61] 5588: CALL DX
[N71] 5589: POP DX
[WN6] 5590: ;Freien Platz auf der Diskette bestimmen
[NR1] 5591: MOV AH,36H
[NB1] 5592: INT 21H
[D71] 5593: MUL CX
[971] 5594: MUL BX
[V94] 5595: ;Long-Integer-Zahl ausgeben
[AN1] 5596: MOV BX,10
[AW1] 5597: XOR CX,CX
[FX1] 5598: MOV SI,AX
[ZV1] 5599: MOV AX,DX
[UN2] 5600: X8DJ: MOV DX,DX
[V61] 5601: DIV BX
[HU1] 5602: XCHG AX,SI
[VU1] 5603: DIV BX
[922] 5604: ;Ziffer auf Stack
[V91] 5605: PUSH DX
[HU1] 5606: XCHG AX,SI
[6U1] 5607: AND SI,SI
[SK1] 5608: LOOPNE X8DJ
[LS1] 5609: AND AX,AX
[SK1] 5610: LOOPNE X8DJ
[3W1] 5611: XOR BX,BX
[YN1] 5612: JC NOTC
[7Y4] 5613: ;Eine Ziffer vom Stack holen
[872] 5614: X8DK: POP AX
[4E1] 5615: OR AL,"0"
[R91] 5616: PUSH CX
[172] 5617: ;und ausgeben
[AG2] 5618: MOV DX,X006-L00
[D61] 5619: CALL DX
[HT1] 5620: POP CX
[RN1] 5621: LOOP X8DK
[724] 5622: ;Abschlußtext ausgeben
[9L1] 5623: CALL X8DL
[3M3] 5624: DB " Bytes free ",13,10,0
[372] 5625: X8DL: POP SI
[CN3] 5626: X8DM: MOV AL,CS:[SI]
[RP1] 5627: AND AL,AL
[DD1] 5628: JE X8DN
[AG2] 5629: MOV DX,X006-L00
[D61] 5630: CALL DX
[X61] 5631: INC SI
[2M1] 5632: JMP X8DM
[232] 5633: X8DN: RET 256
[000] 5634:
[NK4] 5635: ;NAME (Datei umbenennen)
[T92] 5636: L8E: PUSH DS
[SW1] 5637: MOV AX,SS
[XV1] 5638: MOV DS,AX
[1W1] 5639: MOV ES,AX
[5Y1] 5640: MOV SI,SP
[AN1] 5641: ADD SI,261
[G93] 5642: ;Länge von String 1
[PQ3] 5643: MOV BL,DS:[SI-1]
[SN1] 5644: CMP BL,255
[4G1] 5645: JNE X8E0
[RL1] 5646: DEC BL
[DV4] 5647: ;String 1 mit 0 abschließen
[F92] 5648: X8E0: MOV BH,0
[L93] 5649: MOV DS:[SI+BX],BH
[WV1] 5650: MOV DX,SI
[P92] 5651: LEA DI,[SI-256]
[M93] 5652: ;Länge von String 2
[922] 5653: MOV BL,DS:[DI-1]
[SN1] 5654: CMP BL,255
[7G1] 5655: JNE X8E1
[R31] 5656: DEC BL
[FV4] 5657: ;String 2 mit 0 abschließen
[934] 5658: X8E1: MOV DS:[DI+BX],BH
[944] 5659: ;MS-DOS-Funktion "Rename"
[NB1] 5660: MOV AH,56H
[NB1] 5661: INT 21H
[G71] 5662: POP DS
[PT5] 5663: ;Zwei Strings vom Stack entfernen
[C71] 5664: RET 512
[000] 5665:
[GH3] 5666: ;Funktion *STICK*
[7K2] 5667: L8F: MOV BP,SP
[2W3] 5668: ;Joystick-Nummer lesen
[133] 5669: MOV BL,SS:[BP+2]
[CY8] 5670: ;Ungleich Null, dann gespeicherten Wert zurückgeben
[US1] 5671: AND BX,BX
[NGL] 5672: JNE X8F5
[L91] 5673: PUSH BX
[3Y4] 5674: ;Vier Zähler initialisieren
[3W1] 5675: XOR BX,BX
[VU1] 5676: XOR BP,BP
[FX1] 5677: XOR DI,SI
[6U1] 5678: XOR DI,DI
[LW3] 5679: ;Maximal 127 Durchgänge
[DR1] 5680: MOV CX,127
[1K2] 5681: ;Lautesprecher aus
[RG1] 5682: IN AL,61H
[ZU1] 5683: AND AL,0FCH
[F81] 5684: OR AL,1
[US1] 5685: OUT 61H,AL
[8H4] 5686: ;Timer 2 in Modus 2 schalten
[ZW1] 5687: MOV AL,0A4H
[NB1] 5688: OUT 43H,AL
[RE1] 5689: MOV AL,0
[LS1] 5690: OUT 42H,AL
[AS1] 5691: MOV AL,94H
[MS1] 5692: OUT 43H,AL
[4N1] 5693: MOV AL,31
[LS1] 5694: OUT 42H,AL
[015] 5695: ;Joystick-Monoflops starten
[112] 5696: MOV DX,0201H
[0V1] 5697: OUT DX,AL
[HG2] 5698: ;Zustand lesen
[8E2] 5699: X8F0: IN AL,DX
[NL4] 5700: ;Vier Zähler weiterzählen
[LE1] 5701: SHR AL,1
[2B1] 5702: ADC BX,0
[1W1] 5703: SHR AL,1
[U11] 5704: ADC BP,0
[LE1] 5705: SHR AL,1
[R11] 5706: ADC SI,0
[LE1] 5707: SHR AL,1
[V11] 5708: ADC DI,0
[G08] 5709: ;Routine auf 32 Timer-Taktyklen synchronisieren
[RK2] 5710: X8F1: IN AL,42H
[FR1] 5711: AND AL,16
[U11] 5712: JE X8F1
[SK2] 5713: X8F2: IN AL,42H
[FR1] 5714: AND AL,16
[CG1] 5715: JNE X8F2
[8B2] 5716: ;Erneut lesen
[5L1] 5717: LOOP X8F0
[6P0] 5718: STI
[NR4] 5719: ;Werte mit 4 multiplizieren
[5P1] 5720: MOV BX,2
[ZS1] 5721: SHL BX,CL
[RS1] 5722: SHL BP,CL
[MU1] 5723: SHL SI,CL
[SS1] 5724: SHL DI,CL
[5H1] 5725: CALL X8F4
[D92] 5726: X8F3: DW 0,0,0,0
[0N2] 5727: X8F4: DW AX,BX
[DT1] 5728: POP BX
[B94] 5729: ;Zählerstände speichern
[XL2] 5730: MOV CS:[BX],AX
```



```
[322] 5731: MOV CS:[BX+2],BP
[222] 5732: MOV CS:[BX+4],SI
[Y22] 5733: MOV CS:[BX+6],DI
[D71] 5734: POP BX
[CE2] 5735: CALL X8F6
[888] 5736: ;Einen gespeicherten Zählerstand zurückklien
[322] 5737: MOV SI
[DC1] 5738: AND BX,3
[TD1] 5739: SHL BX,1
[Y59] 5740: MOV AX,CS:[SI+BX+OFFSETH X8F3-X8F6]
[8W1] 5741: MOV BP,SP
[H13] 5742: MOV SS:[BP+2],AX
[3P0] 5743: RET
[000] 5744:
[2K3] 5745: ;Funktion *STRIG*
[KF2] 5746: L90: MOV BP,SP
[Y25] 5747: ;Zustand der Joystickknöpfe lesen
[112] 5748: MOV DX,0201H
[GM1] 5749: IN AL,DX
[U74] 5750: ;Ergebnis initialisieren
[HW1] 5751: XOR DX,DX
[YN2] 5752: ;Parameter lesen
[413] 5753: MOV AH,SS:[BP+2]
[2C1] 5754: AND AH,6
[VE1] 5755: JNE X900
[LB1] 5756: ;Knopf A1
[UK1] 5757: TEST AL,10H
[112] 5758: JMP SHORT X903
[362] 5759: X900: CMP AH,4
[YE1] 5760: JNE X901
[MB1] 5761: ;Knopf B1
[XR1] 5762: TEST AL,20H
[112] 5763: JMP SHORT X903
[652] 5764: X901: CMP AH,2
[1F1] 5765: JNE X902
[5D1] 5766: ;Knopf A2
[351] 5767: TEST AL,40H
[112] 5768: JMP SHORT X903
[PB1] 5769: ;Knopf B2
[V72] 5770: X902: TEST AL,80H
[W88] 5771: ;Knopf gedrückt, dann -1 (*true*) zurückgeb
[5V2] 5772: X903: JNE X904
[B41] 5773: DEC DX
[NK3] 5774: X904: MOV SS:[BP+2],DX
[3P0] 5775: RET
[000] 5776:
[4K3] 5777: ;Funktion *EXP*
[K83] 5778: L91: MOV DX,2002-L90
[371] 5779: JMP DX
[000] 5780:
[FH7] 5781: ;Funktion *SQ* (Wurzel aus X ist X ^ .5)
[U22] 5782: L92: MOV BP,SP
[3K1] 5783: ;SQ(R) = 0
[S13] 5784: MOV AX,SS:[BP+4]
[LS1] 5785: AND AX,AX
[L91] 5786: JE X921
[WH3] 5787: ;Vorzeichen sichern
[F91] 5788: PUSH AX
[XH4] 5789: ;Absolutwert von X auf Stack
[NR1] 5790: AND AL,7FH
[F91] 5791: PUSH AX
[FH2] 5792: PUSH SS:[BP+2]
[B72] 5793: ;Logarithmus
[SH2] 5794: MOV DX,200J-L90
[D61] 5795: CALL DX
[Z23] 5796: ;Ergebnis durch 2 teilen
[8W1] 5797: MOV BP,SP
[UX3] 5798: CMP BYTE PTR SS:[BP+3],0
[G91] 5799: JE X920
[8W3] 5800: DEC BYTE PTR SS:[BP+3]
[DP3] 5801: ;Exponentialfunktion
[SA3] 5802: X920: MOV DX,2002-L90
[D61] 5803: CALL DX
[F71] 5804: POP AX
[D71] 5805: POP BX
[HD4] 5806: ;Vorzeichen zurückholen
[H71] 5807: POP CX
[MM1] 5808: AND CL,80H
[TL1] 5809: OR CL,CL
[8W1] 5810: MOV BP,SP
[0X2] 5811: ;Ergebnis sichern
[H13] 5812: MOV SS:[BP+2],AX
[T13] 5813: MOV SS:[BP+4],BX
[0C1] 5814: X921: RET
[000] 5815:
[008] 5816: ;Zuweisung an *DATE$* (amerikanisches Forma
[YF2] 5817: L93: MOV BP,SP
[7B1] 5818: ADD BP,3
[W42] 5819: ;Stringlänge
[G13] 5820: MOV CL,SS:[BP-1]
[VE1] 5821: MOV CH,0
[531] 5822: ;Monat
[NF1] 5823: CALL X931
[3U1] 5824: MOV BH,DL
[XN0] 5825: ;Tag
[RF1] 5826: CALL X931
[7U1] 5827: MOV BL,DL
[4U0] 5828: ;Jahr
[NK2] 5829: CALL X931
[FU3] 5830: ;Verkürztes Format?
[ON1] 5831: CMP DX,100
[UD1] 5832: JAE X930
[S11] 5833: ADD DX,1900
[XK2] 5834: X930: MOV CX,DX
[S11] 5835: MOV DX,BX
[GG6] 5836: ;MS-DOS-Funktion "Setze Systemdatum"
[F91] 5837: MOV AH,2BH
[N81] 5838: INT 21H
[CT1] 5839: RET 256
[V23] 5840: ;Dezimalzahl lesen
[0L2] 5841: X931: XOR DX,DX
[8B2] 5842: X932: CALL X933
[N45] 5843: ;Ist es eine gültige Ziffer?
[XK1] 5844: SUB AL,0
[N91] 5845: JB X934
[8L1] 5846: CMP AL,10
[6E1] 5847: JAE X934
[LE1] 5848: MOV AH,0
[F91] 5849: PUSH AX
[FK5] 5850: ;Alten Wert mit 10 multiplizieren
[6N1] 5851: MOV AX,10
[H71] 5852: MUL DX
[2W1] 5853: MOV DX,AX
[971] 5854: POP AX
[9B4] 5855: ;Neue Ziffer dazuzählen
[LS1] 5856: ADD DX,AX
[GC1] 5857: JMP X932
[L24] 5858: ;Ein Zeichen aus String lesen
[U72] 5859: X933: MOV AL,0
[KK1] 5860: JCXZ X934
[EP2] 5861: MOV AL,SS:[BP]
[051] 5862: INC BP
[741] 5863: DEC CX
[7C1] 5864: X934: RET
[000] 5865:
[4VB] 5866: ;Funktion *COMMAND$* (in GW-BASIC nicht vor
[0K2] 5867: L94: SUB SP,252
[ZG4] 5868: ;Pleatz für String schaffen
[8W1] 5869: MOV BP,SP
[SW1] 5870: MOV AX,SS
[1W1] 5871: MOV ES,AX
[F91] 5872: PUSH DS
[ES1] 5873: MOV SI,128
[ER2] 5874: ;Adresse des PSP
[3H2] 5875: MOV DS,DS:[42]
[K76] 5876: ;Anzahl der Zeichen in der Kommandozeile
```

```
[CV0] 5877: LODSB
[ZU1] 5878: MOV CL,AL
[VE1] 5879: MOV CH,0
[V54] 5880: ;Index des Parameterstrings
[N73] 5881: MOV DI,SS:[BP+254]
[871] 5882: AND DX,DX
[3A1] 5883: JE X945
[453] 5884: X940: LEA DI,[BP+1]
[824] 5885: ;Leerzeichen überlesen
[N82] 5886: X941: CALL X946
[0A1] 5887: JE X944
[XG1] 5888: CMP AL," "
[R91] 5889: JE X941
[RL4] 5890: ;Ein Komma ist Trennzeichen
[DE2] 5891: X942: CMP AL," "
[0A1] 5892: JE X944
[DF6] 5893: ;Buchstaben in Großbuchstaben wandeln
[1W1] 5894: CMP AL,"a"
[N91] 5895: JB X943
[S51] 5896: CMP AL,"z"
[R91] 5897: JA X943
[PL1] 5898: SUB AL,32
[G74] 5899: ;Zeichen in String schreiben
[871] 5900: X943: STOSB
[801] 5901: CALL X946
[0A1] 5902: JE X944
[G35] 5903: ;Leerzeichen als Trennzeichen
[XG1] 5904: CMP AL," "
[0A1] 5905: JE X944
[E50] 5906: ;Schalterzeichen gehört zum nächsten String
[TK1] 5907: CMP AL,"/"
[F91] 5908: JNE X942
[S51] 5909: DEC SI
[CS1] 5910: INC CX
[TP5] 5911: ;Eventuell nächsten String lesen
[P22] 5912: X944: DEC DX
[3F1] 5913: JNE X940
[P22] 5914: LEA DX,[DI-1]
[LZ1] 5915: SUB DX,BP
[HR3] 5916: ;Stringlänge sichern
[DL3] 5917: X945: MOV SS:[BP],DL
[G71] 5918: POP DS
[3M2] 5919: JMP SS:[BP+252]
[113] 5920: ;Ein Zeichen lesen
[W22] 5921: X946: MOV AL,13
[XK1] 5922: JCXZ X947
[741] 5923: DEC CX
[CV0] 5924: LODSB
[XB2] 5925: X947: CMP AL,13
[3P0] 5926: RET
[000] 5927:
[E53] 5928: ;Funktion *SIN*
[E63] 5929: L95: MOV DX,4008-L90
[371] 5930: JMP DX
[000] 5931:
[TH3] 5932: ;Funktion *COS*
[H63] 5933: L96: MOV DX,4007-L90
[371] 5934: JMP DX
[000] 5935:
[PK8] 5936: ;Funktion *TAN* (Tangens X = Sinus X / Cosi
[EC2] 5937: L97: MOV BP,SP
[ED3] 5938: ;Cosinus X berechnen
[NN2] 5939: PUSH SS:[BP+4]
[FN2] 5940: PUSH SS:[BP+2]
[9E2] 5941: MOV DX,4008-L90
[D61] 5942: CALL DX
[X22] 5943: ;Sinus X berechnen
[8W1] 5944: MOV BP,SP
[1P2] 5945: PUSH SS:[BP+8]
[VW2] 5946: PUSH SS:[BP+6]
[7E2] 5947: MOV DX,4007-L90
[D61] 5948: CALL DX
[W64] 5949: ;Komponenten dividieren
[XW2] 5950: MOV DX,DIV5-L90
[D61] 5951: CALL DX
[0L4] 5952: ;Ergebnis zurückschreiben
[8W1] 5953: MOV BP,SP
[NL2] 5954: POP SS:[BP+6]
[VL2] 5955: POP SS:[BP+8]
[3P0] 5956: RET
[000] 5957:
[KH3] 5958: ;Funktion *ATN*
[KG2] 5959: L98: MOV BP,SP
[S13] 5960: MOV AX,SS:[BP+4]
[B73] 5961: ;Absolutwert von X
[NR1] 5962: AND AL,7FH
[901] 5963: X < 1?
[S11] 5964: CMP AH,81H
[L91] 5965: JB X980
[R22] 5966: ;Nein, dann X = 1 / X
[B12] 5967: MOV DX,8100H
[L91] 5968: PUSH BX
[3W1] 5969: XOR BX,BX
[L91] 5970: PUSH BX
[F91] 5971: PUSH AX
[FN2] 5972: PUSH SS:[BP+2]
[KM2] 5973: MOV DX,DIV5-L90
[D61] 5974: CALL DX
[L22] 5975: JMP SHORT X981
[N42] 5976: X980: PUSH AX
[FN2] 5977: PUSH SS:[BP+2]
[NK2] 5978: X981: MOV BP,SP
[CT1] 5979: X1 = X * X
[FN2] 5980: PUSH SS:[BP+2]
[982] 5981: PUSH SS:[BP]
[FN2] 5982: PUSH SS:[BP+2]
[982] 5983: PUSH SS:[BP]
[L22] 5984: MOV DX,MUL5-L90
[D61] 5985: CALL DX
[F63] 5986: ;X2 = X1 x -6.847939E-4
[G22] 5987: MOV AX,76B3H
[F91] 5988: PUSH AX
[932] 5989: MOV AX,83BDH
[F91] 5990: PUSH AX
[8W1] 5991: MOV BP,SP
[VW2] 5992: PUSH SS:[BP+6]
[NW2] 5993: PUSH SS:[BP+4]
[L22] 5994: MOV DX,MUL5-L90
[D61] 5995: CALL DX
[XU2] 5996: ;X2 = X2 + 4.859942E-3
[F32] 5997: MOV AX,791EH
[F91] 5998: PUSH AX
[L82] 5999: MOV AX,0F4A6H
[F91] 6000: PUSH AX
[3L2] 6001: MOV DX,ADDS-L90
[D61] 6002: CALL DX
[MU1] 6003: ;X2 = X2 x X1
[8W1] 6004: MOV BP,SP
[VW2] 6005: PUSH SS:[BP+6]
[NW2] 6006: PUSH SS:[BP+4]
[L22] 6007: MOV DX,MUL5-L90
[D61] 6008: CALL DX
[MG2] 6009: ;X2 = X2 - 0.61117
[F32] 6010: MOV AX,7B83H
[F91] 6011: PUSH AX
[KB2] 6012: MOV AX,0FCB0H
[F91] 6013: PUSH AX
[3L2] 6014: MOV DX,ADDS-L90
[D61] 6015: CALL DX
[MU1] 6016: ;X2 = X2 x X1
[8W1] 6017: MOV BP,SP
[VW2] 6018: PUSH SS:[BP+6]
[NW2] 6019: PUSH SS:[BP+4]
[L22] 6020: MOV DX,MUL5-L90
[D61] 6021: CALL DX
[G22] 6022: ;X2 = X2 + 0.342096
[K42] 6023: MOV AX,7C9CH
[F91] 6024: PUSH AX
[U32] 6025: MOV AX,1F67H
[F91] 6026: PUSH AX
```

```
[3L2] 6027: MOV DX,ADDS-L90
[D61] 6028: CALL DX
[MU1] 6029: ;X2 = X2 x X1
[8W1] 6030: MOV BP,SP
[VW2] 6031: PUSH SS:[BP+6]
[NW2] 6032: PUSH SS:[BP+4]
[L22] 6033: MOV DX,MUL5-L90
[D61] 6034: CALL DX
[5H2] 6035: ;X2 = X2 - 0.542791
[A52] 6036: MOV AX,7CDEH
[F91] 6037: PUSH AX
[X22] 6038: MOV AX,53CBH
[F91] 6039: PUSH AX
[3L2] 6040: MOV DX,ADDS-L90
[D61] 6041: CALL DX
[MU1] 6042: ;X2 = X2 x X1
[8W1] 6043: MOV BP,SP
[VW2] 6044: PUSH SS:[BP+6]
[NW2] 6045: PUSH SS:[BP+4]
[L22] 6046: MOV DX,MUL5-L90
[D61] 6047: CALL DX
[NG2] 6048: ;X2 = X2 + 0.724571
[T32] 6049: MOV AX,7D14H
[F91] 6050: PUSH AX
[M12] 6051: MOV AX,6470H
[F91] 6052: PUSH AX
[3L2] 6053: MOV DX,ADDS-L90
[D61] 6054: CALL DX
[MU1] 6055: ;X2 = X2 x X1
[8W1] 6056: MOV BP,SP
[VW2] 6057: PUSH SS:[BP+6]
[NW2] 6058: PUSH SS:[BP+4]
[L22] 6059: MOV DX,MUL5-L90
[D61] 6060: CALL DX
[2K2] 6061: ;X2 = X2 - 0.898019
[G42] 6062: MOV AX,7DB7H
[F91] 6063: PUSH AX
[N2A] 6064: MOV AX,0E51H
[F91] 6065: PUSH AX
[3L2] 6066: MOV DX,ADDS-L90
[D61] 6067: CALL DX
[MU1] 6068: ;X2 = X2 x X1
[8W1] 6069: MOV BP,SP
[VW2] 6070: PUSH SS:[BP+6]
[NW2] 6071: PUSH SS:[BP+4]
[L22] 6072: MOV DX,MUL5-L90
[D61] 6073: CALL DX
[V22] 6074: ;X2 = X2 + 1.109324
[W32] 6075: MOV AX,7D63H
[F91] 6076: PUSH AX
[C12] 6077: MOV AX,3088H
[F91] 6078: PUSH AX
[3L2] 6079: MOV DX,ADDS-L90
[D61] 6080: CALL DX
[MU1] 6081: ;X2 = X2 x X1
[8W1] 6082: MOV BP,SP
[VW2] 6083: PUSH SS:[BP+6]
[NW2] 6084: PUSH SS:[BP+4]
[L22] 6085: MOV DX,MUL5-L90
[D61] 6086: CALL DX
[PH2] 6087: ;X2 = X2 - 1.428398
[L42] 6088: MOV AX,7E92H
[F91] 6089: PUSH AX
[222] 6090: MOV AX,4499H
[F91] 6091: PUSH AX
[3L2] 6092: MOV DX,ADDS-L90
[D61] 6093: CALL DX
[MU1] 6094: ;X2 = X2 x X1
[8W1] 6095: MOV BP,SP
[VW2] 6096: PUSH SS:[BP+6]
[NW2] 6097: PUSH SS:[BP+4]
[L22] 6098: MOV DX,MUL5-L90
[D61] 6099: CALL DX
[CK2] 6100: ;X2 = X2 + 1.359991
[V42] 6101: MOV AX,7E4CH
[F91] 6102: PUSH AX
[RA2] 6103: MOV AX,0CC91H
[F91] 6104: PUSH AX
[3L2] 6105: MOV DX,ADDS-L90
[D61] 6106: CALL DX
[MU1] 6107: ;X2 = X2 x X1
[8W1] 6108: MOV BP,SP
[VW2] 6109: PUSH SS:[BP+6]
[NW2] 6110: PUSH SS:[BP+4]
[L22] 6111: MOV DX,MUL5-L90
[D61] 6112: CALL DX
[G02] 6113: ;X2 = X2 - 3.333333
[B52] 6114: MOV AX,7FAAH
[F91] 6115: PUSH AX
[DC2] 6116: MOV AX,9AAAH
[F91] 6117: PUSH AX
[3L2] 6118: MOV DX,ADDS-L90
[D61] 6119: CALL DX
[MU1] 6120: ;X2 = X2 x X1
[8W1] 6121: MOV BP,SP
[VW2] 6122: PUSH SS:[BP+6]
[NW2] 6123: PUSH SS:[BP+4]
[L22] 6124: MOV DX,MUL5-L90
[D61] 6125: CALL DX
[4K1] 6126: ;X2 = X2 + 1
[712] 6127: MOV AX,8100H
[F91] 6128: PUSH AX
[WV1] 6129: XOR AX,AX
[F91] 6130: PUSH AX
[3L2] 6131: MOV DX,ADDS-L90
[D61] 6132: CALL DX
[ER1] 6133: ;X1 = X2 x X
[8W1] 6134: MOV BP,SP
[F22] 6135: POP SS:[BP+4]
[NL2] 6136: POP SS:[BP+6]
[L22] 6137: MOV DX,MUL5-L90
[D61] 6138: CALL DX
[4N1] 6139: ;War X < 1?
[8W1] 6140: MOV BP,SP
[P74] 6141: CMP BYTE PTR SS:[BP+9],81H
[W91] 6142: JB X983
[3P3] 6143: ;Nein, dann X1 = P1/2 - X1
[CM2] 6144: CMP BYTE PTR SS:[BP+3],0
[ZAL] 6145: JE X982
[E84] 6146: XOR BYTE PTR SS:[BP+2],80H
[TV2] 6147: X982: MOV AX,8149H
[F91] 6148: PUSH AX
[742] 6149: MOV AX,0FDAH
[F91] 6150: PUSH AX
[3L2] 6151: MOV DX,ADDS-L90
[D61] 6152: CALL DX
[8W1] 6153: MOV BP,SP
[P54] 6154: ;Vorzeichen von X holen
[3M5] 6155: X983: TEST BYTE PTR SS:[BP+8],80H
[8A1] 6156: JE X984
[UX3] 6157: CMP BYTE PTR SS:[BP+3],0
[8A1] 6158: JE X984
[E84] 6159: XOR BYTE PTR SS:[BP+2],80H
[RV3] 6160: ;Ergebnis zurückgeben
[AB3] 6161: X984: POP SS:[BP+6]
[V12] 6162: POP SS:[BP+8]
[3P0] 6163: RET
[000] 6164:
[GC1] 6165: L99 = $
[000] 6166:
[1R1] 6167: CODE ENDS
[4M0] 6168: END
```

Listing 2. Der Quelltext der SBasic-Laufzeitbibliothek »sbasic.asm«


```
[LU5] 1: 36 01 8D 0C A2 0C B5 0C C8 0C DB 0C EE 0C 01 0D
[QN5] 2: 14 0D 1F 0D 2A 0D 35 0D 4D 0D 4B 0D 59 0D 69 0D
[7X5] 3: 79 0D 7F 0D 85 0D A6 0D C4 0D DD 0D FD 0D 3E 0E
[QU5] 4: 47 0E 76 0E 89 0E 9C 0E B8 0E DC 0E 24 0F 42 0F
[WF5] 5: C9 10 D6 10 0E 11 5C 11 6C 11 FE 11 0C 12 18 12
[GD5] 6: 30 12 5E 12 71 12 84 12 97 12 AA 12 BD 12 D0 12
[HC5] 7: FB 12 05 13 47 13 63 13 70 13 96 13 A6 13 B8 13
[OH5] 8: CD 13 E1 13 0A 14 44 14 68 14 8A 14 98 14 D9 14
[8G5] 9: 15 15 45 15 D9 15 0C 16 3F 16 9F 16 CE 16 5A 18
[9K5] 10: 61 18 66 18 A0 18 A5 18 AA 18 AF 18 E9 18 09 19
[8K5] 11: 2F 19 34 19 5C 19 71 19 86 19 9B 19 B0 19 C5 19
[4T5] 12: DA 19 EA 19 03 1A 75 1A 90 1A A4 1A B6 1A D2 1A
[NX5] 13: E5 1A F1 1A FD 1A 15 1B 26 1B AA 1C D8 1C EA 1C
[6W5] 14: FF 1C 1B 1D 96 1D CB 1D 1C 1E 64 1E 78 1E 95 1E
[KS5] 15: D5 1E 07 1F 13 1F 37 1F 65 1F 71 1F 86 1F CB 1F
[SP5] 16: DC 1F F9 1F 27 20 B3 20 C1 20 8E 21 AA 21 CF 21
[V85] 17: F1 21 13 22 35 22 40 22 46 22 65 22 E5 22 66 23
[ZD5] 18: 74 23 80 23 0A 26 16 26 1D 26 3F 26 F8 27 2D 28
[YS5] 19: BD 28 EC 28 F1 28 27 29 78 29 DA 29 DF 29 E4 29
[LZ5] 20: 0C 2A AB 2B FF FF 8C DD 8C CA 03 C2 8E D8 8E C0
[KM5] 21: 23 DB 74 11 33 C0 33 FF B9 08 00 F3 AB 8C C0 40
[BK5] 22: 8E C0 4B 75 EF 89 2E 2A 00 C6 06 00 00 FF C6 06
[AH5] 23: 0A 00 04 C6 06 09 00 07 B4 0F CD 10 FE CC 88 26
[9K5] 24: 0B 00 C6 06 0D 07 8C 1E 19 00 B4 03 CD 10 89
[9T5] 25: 0E 0F 0C 07 06 15 00 FF FF 80 FE 18 72 0B B8 0A
[YL5] 26: 0E CD 10 B6 17 B4 02 CD 10 B9 00 1F B4 01 CD 10
[CN5] 27: BA 07 07 B4 0F CD 10 3C 07 75 03 BA 0C 0B 89 16
[YK5] 28: 2A 00 07 06 26 00 A4 8E C7 06 28 00 1F 7F C7 06
[Q95] 29: 2C 00 30 04 C7 06 2E 00 78 07 C3 00 00 00 00 00
[QZ4] 30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[QZ4] 31: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[QZ4] 32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[QZ4] 33: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[QZ4] 34: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[QZ4] 35: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[QZ4] 36: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[QZ4] 37: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[ZC5] 38: 00 00 00 00 00 00 00 00 50 52 BA FD 03 EC D0 E8 73
[FT5] 39: 1F B2 F8 EC 53 2E 8A 1E 1D 01 43 83 E3 3F 2E 3A
[5L5] 40: 1E 1E 01 74 0A 2E 88 87 95 00 2E 88 1E 1D 01 5B
[1R5] 41: B0 64 E6 20 5A 58 CF 50 52 BA FD 02 EC D0 E8 73
[NT5] 42: 1F B2 F8 EC 53 2E 8A 1E 1F 01 43 83 E3 3F 2E 3A
[PK5] 43: 1E 20 01 74 0A 2E 88 87 95 00 2E 88 1E 1F 01 5B
[3S5] 44: B0 63 E6 20 5A 58 CF 3C 0D 74 09 3C 0A 74 09 FE
[SD5] 45: 47 04 EB 04 C6 47 04 00 8A 1F B9 01 00 A2 0E 0E
[EP5] 46: BA 0E 00 B4 40 CD 21 5B C3 53 8A 0F 80 F9 FD 72
[LW5] 47: D6 50 80 F9 FE 75 0E BA FD 03 EC A8 20 74 FB B2
[8Z5] 48: F8 58 EE 5B C3 80 F9 FD 75 0E BA FD 02 EC A8 20
[4S5] 49: 74 FB B2 F8 58 EE 5B C3 80 F9 FD 75 0E BA FD 02 EC A8 20
[XK5] 50: 59 8A 1E 09 00 80 F9 07 75 07 B8 07 0E CD 10 5B
[WR5] 51: C3 80 F9 09 75 2E 88 CC 00 B4 03 CD 10 8A C2 24
[GD5] 52: 07 B9 08 00 2A C8 E8 20 09 CD 10 C6 06 0A 00 00
[CM5] 53: 02 D1 3A 16 0B 00 76 06 FE CA FE 0E 0A 00 B4 02
[YM5] 54: CD 10 5B C3 80 F9 0A 74 05 80 F9 0D 75 05 E8 A1
[CG5] 55: 00 5B C3 80 F9 0B 75 11 B2 00 8A 36 0C 00 B4 02
[SM5] 56: CD 10 C6 06 0A 00 00 5B C3 80 F9 0C 75 08 B8 00
[UL5] 57: 06 E8 9C 00 EB E2 80 5F 1C 75 12 B4 03 CD 10 3A
[BR5] 58: 16 0B 00 74 04 FE C2 EB 15 B2 00 EB 09 80 F9 1F
[7M5] 59: 75 17 B4 03 CD 10 3A 36 0D 00 74 0B FE C6 B4 02
[YM5] 60: CD 10 C6 06 0A 00 00 5B C3 80 F9 1D 75 13 B4 03
[AX5] 61: CD 10 80 FA 00 74 1D FE CA EB E3 8A 16 0B 00 74
[CN5] 62: 09 80 F9 1E 75 0E B4 03 CD 10 3A 36 0C 00 B4 D7
[ZT5] 63: FE CE EB CA E8 0E 00 8A C1 B4 09 B9 01 00 CD 10
[ZE5] 64: E8 46 00 5B C3 80 F9 0A 74 05 80 F9 0D 75 05 E8 A1
[NG5] 65: 74 36 51 B2 00 3A 36 0D 00 75 08 B8 01 06 E8 0F
[VK5] 66: 00 EB 02 FE C6 B4 02 CD 10 C6 06 0A 00 00 59 C3
[RM5] 67: 53 51 52 8A FB B1 00 8A 2E 0C 00 8A 16 0B 00 8A
[MM5] 68: 36 0D 00 CD 10 5A 59 5B C3 B4 03 CD 10 3A 16 0B
[ML5] 69: 00 74 0C FE C2 B4 02 CD 10 C6 06 0A 00 00 C3 C6
[RN5] 70: 06 0A 00 FF C3 B4 00 1A B8 3E 17 00 23 FF 74 0F 2E
[BN5] 71: 8A 05 47 3C 1A 75 03 2E 8B 3D 89 3E 17 00 C3 23
[1U5] 72: DB 78 E2 8A 07 3C FF 74 71 53 3C FE 75 20 2E 8A
[SN5] 73: 1E 1E 01 2E 3A 1E 1D 01 74 F4 FE C3 80 E3 3F 2E
[UN5] 74: 8A 87 95 00 2E 88 1E 1E 01 5B 88 7F 10 C3 3C FD
[2M5] 75: 75 20 2E 8A 1E 20 01 2E 3A 1E 1F 01 74 F4 FE C3
[GL5] 76: 80 E3 3F 2E 8A 87 D5 00 2E 88 1E 20 01 5B 88 7F
[DM5] 77: 10 C3 5B 8A 4F 10 51 80 F9 1A 74 17 53 8A D8 B9
[RL5] 78: 01 00 BA 0E 00 B4 3F CD 21 5B 72 07 23 C0 A0 0E
[3H5] 79: 00 75 02 B0 1A 88 47 10 58 C3 A0 0E 03 3C FF 75
[CK5] 80: 05 B4 08 CD 21 C3 83 3E 15 00 FF 74 28 3C 00 75
[SL5] 81: 03 B0 1A C3 FE 0E 00 53 B4 0F CD 10 B4 08 CD
[XG5] 82: 10 50 B4 03 CD 10 3A 16 0B 00 73 06 FE C2 B4 02
[FT5] 83: CD 10 58 5B C3 53 B4 0F CD 10 B4 03 CD 10 89 0E
[KP5] 84: 15 00 52 33 FF 3A 16 0B 00 76 0A 16 0B 00 B4
[TK5] 85: 02 CD 10 8B 0E 21 50 00 23 FF 74 02 B5 04 B4 01 CD
[EG5] 86: 10 B4 08 CD 21 50 B9 00 1F B4 01 CD 10 58 3C 20
[CN5] 87: 72 44 8A 1E 09 00 23 FF 74 31 50 8A CA 8A 16 0B
[LS5] 88: 00 FE CA 78 1F 3A D1 72 1B B4 02 CD 10 B4 08 CD
[2H5] 89: 10 FE C2 B4 02 CD 10 51 B9 01 00 B4 09 CD 10 59
[BV5] 90: FE CA EB DD 8A D1 B4 02 CD 10 58 B9 01 00 B4 09
[KT5] 91: CD 10 FE C2 EB 8F 3C 00 75 4D B4 08 CD 21 3C 4B
[TP5] 92: 75 0B 58 50 3A D0 76 02 FE CA E9 78 FF 3C 4D 75
[LU5] 93: 04 FE C2 EB F5 3C 47 75 04 5A 52 EB ED 3C 4F 75
[UH5] 94: 1A 8A 16 0B 00 59 51 B4 02 CD 10 B4 08 CD 10 3C
[RW5] 95: 20 75 DE 3A D1 74 D3 FE CA EB EC 3C 53 74 14 3C
[TR5] 96: 52 75 C7 F7 D7 EB C3 3C 08 75 EC 59 51 3A D1 76
```

```
[2L5] 97: 35 FE CA 52 8A 1E 09 00 3A 16 0B 00 73 1B FE C2
[6P5] 98: B4 02 CD 10 B4 08 CD 10 FE CA B4 02 CD 10 B9 01
[ZS5] 99: 00 B4 09 CD 10 FE C2 EB DF B4 02 CD 10 B8 20 09
[ZR5] 100: B9 01 00 CD 10 5A E9 FC FE 3C 0D 74 07 3C 0A 74
[8R5] 101: F5 E9 1E FF 59 8A 16 0B 00 B4 02 CD 10 B4 08 CD
[4S5] 102: 10 3C 20 75 08 FE CA 78 04 3A D1 73 EC FE C2 2A
[LP5] 103: D1 88 16 0E 00 8A D1 B4 02 CD 10 5B E9 7B FE 80
[NN5] 104: 3F FF 75 13 B0 0D E8 60 FC 8B 0E 15 00 B4 01 CD
[AL5] 105: 10 C7 06 15 00 FF FF C2 02 00 8B EC 83 C5 04 8A
[FM5] 106: 8E 00 01 8A 6E 00 33 FF E3 1C 80 F9 00 74 14 80
[6P5] 107: FD 00 74 15 47 FE C9 FE CD 8A 83 00 01 3A 03 74
[GH5] 108: E7 77 06 B0 02 C3 B0 01 C3 B0 04 C3 8B EC B2 00
[4N5] 109: 8A 76 06 8B 76 07 8A 5E 09 B0 00 8A 66 02 8B 7E
[1N5] 110: 03 8A 7E 05 3A DF 73 05 92 87 F7 86 DF 80 FF 00
[QR5] 111: 74 6A 8A CB 2A CF 80 F9 18 77 61 B5 00 56 33 F7
[8S5] 112: 9C 8B 76 FE 81 CE 00 80 81 CF 00 80 83 06 D1 EF
[WS5] 113: D1 D8 E2 FA 9D 78 0E 03 D0 13 F7 73 39 D1 DE D1
[5T5] 114: DA FE C3 EB 31 2B D0 17 F7 72 08 75 14 23 C2 75
[SN5] 115: 10 EB 1A F7 D2 83 C2 01 F7 D6 83 D6 00 80 76 FF
[HR5] 116: 80 23 F6 78 11 D1 E2 D1 D6 FE CB 75 F4 33 F6 33
[SM5] 117: D2 B3 00 58 EB 06 58 0D FF 7C 23 F0 88 76 06 89
[BL5] 118: 76 07 88 5E 09 C2 04 00 8B EC 80 7E 09 00 74 5B
[HL5] 119: 80 7E 05 00 74 55 8A 5E 02 8A 7E 06 8A C3 F6 E7
[9N5] 120: 8B F8 8B 76 03 8B 6E 07 8B C6 33 C5 0D FF 7F 50
[ZM5] 121: 81 CE 00 80 81 CD 00 80 8B C6 F7 E5 8B C8 87 F2
[ML5] 122: B0 00 8A E7 F7 E2 03 F8 13 CA 83 D6 00 80 B0 00 8A
[8R5] 123: E3 F7 E5 03 F8 13 CA 83 D6 00 80 8B C6 F7 E5 8B C8 87 F2
[6F5] 124: 8A 7E 09 81 F3 80 80 02 DF 71 0B 33 C0 89 46 06
[DL5] 125: 89 46 08 C2 04 00 80 F3 80 23 F6 78 0A D1 E7 D1
[PN5] 126: D1 D1 D6 FE CB 75 F2 81 C1 00 80 83 D6 00 73 06
[YS5] 127: D1 DE D1 D9 FE C3 23 F0 88 6E 06 89 76 07 88 5E
[EE5] 128: 09 C2 04 00 8B EC 80 7E 09 00 74 76 80 7E 05 00
[TG5] 129: 74 73 8B 76 06 8A 5E 08 B7 08 B7 7E 02 8A 56 04
[PK5] 130: B6 00 8A C3 32 C2 0C 7F 50 80 CB 80 80 80 B1
[BR5] 131: 19 33 ED B0 00 2B F7 1B DA 73 04 03 F7 13 DA F5
[HT5] 132: D1 D5 D0 D0 D1 E6 D1 D3 FE C9 22 C0 79 E7 2B F7
[ER5] 133: 1B DA F5 83 D5 00 14 00 73 06 D0 D8 D1 DD FE C1
[5R5] 134: 5B 8B D5 8B EC 22 C3 89 56 06 88 46 08 8A 46 09
[NC5] 135: 8A 66 05 35 80 80 02 C1 2A C4 70 09 35 80 00 88
[1D5] 136: 46 09 C2 04 00 33 C0 89 46 06 89 46 08 C2 04 00
[LM5] 137: 8B EC 8A 5E 05 22 DB 74 0F B0 80 84 46 04 74 09
[BB5] 138: 33 C0 89 46 02 89 46 04 C3 88 46 05 81 7E 04 35
[TK5] 139: 80 75 05 81 7E 02 F4 04 73 05 FE 46 05 FE CB 53
[RS5] 140: FF 76 04 FF 76 02 B8 80 81 50 33 C0 50 E8 FC FD
[1T5] 141: FF 76 10 FF 76 0E B8 00 81 50 33 C0 50 E8 EC FD
[YK5] 142: E8 21 FF 8B 46 08 89 46 10 50 8B 46 06 89 46 0E
[PN5] 143: 50 E8 74 FE 8B 46 08 8B 5E 0A C3 50 53 05 E8 67
[TX5] 144: FE B8 5E 7F 50 B8 4D 57 50 E8 C0 FD FF 76 0C FF
[MR5] 145: 76 0A E8 53 FE B8 13 80 50 B8 08 9B 50 E8 AC FD
[GU5] 146: FF 76 0C FF 76 0A E8 3F FE B8 76 80 50 E8 93 38
[DU5] 147: 50 E8 98 FD E8 31 FE B8 38 82 50 B8 3B AA 50 E8
[KV5] 148: 8A FD FF 76 10 FF 76 0E E8 1D FE 8A 46 04 B4 00
[CG5] 149: 34 80 74 17 8A D8 79 02 F6 D8 B4 88 22 C0 78 06
[VV5] 150: D0 0E FE CC EB F6 80 CB 7F 22 C3 50 33 C0 50 E8
[2U5] 151: 5A FD B8 31 80 50 B8 18 72 50 E8 EB FD 8F 46 0E
[TM5] 152: 8F 46 10 58 C3 8B EC B2 00 8B 76 06 8B 46 08 8B
[OT5] 153: 7E 02 8B 5E 04 8A C8 8A EB 32 E9 78 19 0C 80 80
[TH5] 154: CB 80 3B C3 75 04 3B F7 74 0B 77 03 80 F1 80 D0
[SV5] 155: E1 72 07 FE C2 C3 D0 E0 73 F9 FE CA C3 53 8A 1F
[V55] 156: 80 FB FF 74 4F 80 FB 05 72 4A 33 C0 8E C0 80 FB
[9S5] 157: FE 75 1C BA F9 03 EE A4 21 0C 10 E6 21 2E A1 15
[SC5] 158: 01 26 A3 30 00 2E A1 17 01 26 A3 32 00 EB 25 80
[5S5] 159: FB FD 75 1C BA F9 02 EE E4 21 0C 08 E6 21 2E A1
[XG5] 160: 19 01 26 A3 2C 00 2E A1 1B 01 26 A3 2E 00 EB 04
[HP5] 161: B4 3E CD 21 5B C3 07 B8 38 81 50 B8 3B AA 50 E8
[6M5] 162: 46 FD 33 D2 33 C0 8B 5E 08 80 FF 81 72 24 8A E3
[1N5] 163: 80 CC 80 8A 46 07 8A 56 06 08 80 FF 90 73 09 D1 E8
[XS5] 164: 80 D6 00 FE C7 EB F2 22 DB 79 07 F7 D8 23 D2 74
[NH5] 165: 01 48 50 FF 76 0E FF 76 06 B7 06 B7 00 23 C0 74 0C 79
[PS5] 166: 02 F7 D8 B7 90 FE CF D1 E0 79 F7 00 83 80 80 E4
[XU5] 167: 7F 0A E3 8A DC 8A E0 B0 00 00 F3 80 53 50 E8 4B
[YL5] 168: FC B8 34 71 50 B8 3C 58 50 FF 76 0F F7 76 06 E8
[3X5] 169: D6 FC B8 16 74 50 B8 B7 7E 50 E8 2F FC FF 76 0C E8
[805] 170: FF 76 0A E8 C2 FC B8 2F 77 50 B8 DF E8 50 E8 1B
[VY5] 171: FC FF 76 0C FF 76 0A E8 AE FC B8 1D 7A 50 B8 17
[EV5] 172: 84 50 E8 07 FC FF 76 0C FF 76 0A E8 9A FC B8 63
[6W5] 173: 7C 50 B8 59 59 50 E8 F3 FB 76 0A E8 7F 76 06 E8
[B16] 174: 86 FC B8 75 7E 50 B8 E9 FD 50 E8 DF FB FF 76 0C E8
[WR5] 175: FF 76 0A E8 72 FC B8 31 80 50 B8 18 72 50 E8 CB
[SS5] 176: FB FF 76 0C FF 76 0A E8 5E FC B8 00 81 50 33 C0
[DN5] 177: 50 E8 B8 FB 58 5B 59 59 50 2F 75 04 33 C0 33
[VP5] 178: DB 89 46 10 89 5E 12 06 C3 8B EC 8B 46 04 80 FC
[UM5] 179: 98 73 50 22 C0 79 1C FF 76 0A FF 76 02 B8 FF 80
[OU5] 180: 50 B8 FF FF 50 E8 84 FB 8B EC 83 C5 04 8F 46 02
[NSS] 181: 8F 46 04 B8 FF FF B3 FF B1 90 2A E4 05 73 07 80
[BK5] 182: C1 08 D2 E0 EB 17 B0 00 80 E9 08 73 07 80 C1 08
[KH5] 183: D2 E4 EB 09 B4 00 80 F9 08 73 09 D2 E3 21 46 02
[4F5] 184: 20 5E 04 C3 89 46 02 89 46 04 C3 07 B8 49 81 50
[5N5] 185: B8 DA 0F 50 E8 35 FB 06 07 B8 22 7E 50 B8 81 F9
[8K5] 186: 50 E8 C4 FB 8B EC FF 76 02 F7 76 0C E8 7A FF 8B
[CM5] 187: EC 80 7E 03 00 74 04 80 76 02 80 E8 0E FB 8B EC
[EC5] 188: 80 7E 03 00 74 04 80 76 02 80 B8 00 7F 50 33 C0
[SN5] 189: 50 E8 F8 FA 8B EC F6 46 02 80 74 19 B8 00 80 50
[WN5] 190: 33 C0 50 E8 E6 FA 8B EC 80 66 02 7F B8 80 7F 50
[CD5] 191: 33 C0 50 EB 11 80 7E 03 00 74 04 80 76 02 80 B8
[LU5] 192: 00 7F 50 33 C0 50 E8 C3 FA 8B EC FF 76 02 FF 76
```



```
[V55] 193: 00 FF 76 02 FF 76 00 E8 4E FB B8 E6 84 50 B8 2D
[HW5] 194: 1A 50 8B EC FF 76 06 FF 76 04 E8 3B FB B8 28 86
[HS5] 195: 50 B8 FB 07 50 E8 94 FA 8B EC FF 76 06 FF 76 04
[LM5] 196: E8 25 FB B8 99 87 50 B8 99 68 50 E8 7E FA 8B EC
[FU5] 197: FF 76 06 FF 76 04 E8 0F FB B8 23 87 50 B8 DF 35
[OW5] 198: 50 E8 68 FA 8B EC FF 76 06 FF 76 04 E8 F9 FA B8
[MW5] 199: A5 86 50 B8 E7 5D 50 E8 52 FA 8B EC FF 76 06 FF
[6T5] 200: 76 04 E8 E3 FA B8 49 83 50 B8 DA 0F 50 E8 3C FA
[FM5] 201: 8B EC 8F 46 04 8F 46 06 E8 CD FA 06 C3 B8 03 00
[TV5] 202: E8 1A FD 4B 75 FA 8B 0E 0F 00 B4 01 CD 10 B4 4C
[XL5] 203: CD 21 8B EC 33 D2 8B 46 04 3B 46 02 7E 01 4A 89
[NH5] 204: 56 04 C2 02 00 8B EC 33 D2 8B 46 04 3B 46 02 7D
[XF5] 205: 01 4A 89 56 04 C2 02 00 8B EC 33 D2 8B 46 04 3B
[GK5] 206: 46 02 75 01 4A 89 56 04 C2 02 00 8B EC 33 D2 8B
[CG5] 207: 46 04 3B 46 02 7C 01 4A 89 56 04 C2 02 00 8B EC
[RE5] 208: 33 D2 8B 46 04 3B 46 02 7F 01 4A 89 56 04 C2 02
[MP5] 209: 00 8B EC 33 D2 8B 46 04 3B 46 02 74 01 4A 89 56
[UC5] 210: 04 C2 02 00 8B EC 8B 46 02 01 46 04 C2 02 00 8B
[6H5] 211: EC 8B 46 02 29 46 04 C2 02 00 8B EC 8B 46 02 21
[VD5] 212: 46 04 C2 02 00 8B EC 8B 46 02 09 46 04 C2 02 00
[DH5] 213: 8B EC 8B 46 02 31 46 04 C2 02 00 8B EC 8B 46 04
[HK5] 214: F7 66 02 89 46 04 C2 02 00 8B EC 8B 46 04 33 D2
[GL5] 215: F7 7E 02 89 46 04 C2 02 00 8B EC 8B 46 04 33 D2
[AP5] 216: F7 7E 02 89 56 04 C2 02 00 8B EC F7 56 02 C3 8B
[WS5] 217: EC F7 5E 02 C3 8B EC 8B 5E 00 8C DA C5 76 02 81
[DQ6] 218: EC FA 00 8B EC 8C 8D 8E 0C FC AC AA C8 B5 00
[FV5] 219: F3 A4 8E DA FF E3 8B F4 C3 C6 02 1E 8C D0 8E D8
[3U5] 220: 33 FF 36 8E 8A 00 01 FC AC AA 8A C8 B5 00 F3 A4
[WP5] 221: 1F C2 02 01 8B EC 8B 5E 02 B0 0D BA A3 01 FF D2
[9T5] 222: 80 3F FF 74 07 B0 0A BA A3 01 FF D2 C3 8B EC 83
[LR5] 223: C5 02 8B 9E 00 01 8A 4E 00 B5 00 E3 0D 45 8A 46
[5R5] 224: 00 51 BA A3 01 FF D2 59 E2 F3 C2 00 01 81 EC FC
[KP5] 225: 00 8B EC 8B 86 FE 00 B3 20 23 C0 79 04 F7 D8 B3
[LM5] 226: 2D 88 5E 01 BB 0A 00 33 C9 33 D2 F7 F3 80 CA 30
[WL5] 227: 52 23 C0 E0 F4 F7 D9 8A D9 BF 02 00 58 88 03 47
[BV5] 228: E2 FA C6 03 20 FE C3 88 5E 00 FF A6 FC 00 33 DB
[8W5] 229: B0 0C BA A3 01 FF E2 8B F4 83 C6 02 1E 8C D0 8E
[LT5] 230: D8 8E C0 8A 8C 00 01 B5 01 8B F9 AC 03 FE B5 00
[EN5] 231: F7 D9 81 C1 FF 00 3A C1 73 02 8A C8 00 8C FF 00
[VR5] 232: F3 A4 1F C2 00 01 81 EC FC 00 8B EC B0 01 8A A6
[ZN5] 233: FE 00 89 46 00 FF A6 FC 00 8B EC 33 D2 39 56 02
[NL5] 234: 74 06 7F 03 4A EB 01 42 89 56 02 C3 8B EC B4 0F
[3F5] 235: CD 10 8A 56 02 8A 76 04 81 EA 01 01 B4 02 CD 10
[2C5] 236: C6 06 0A 00 00 C2 04 00 8B EC 8A 46 04 8A 66 02
[6M5] 237: B1 04 D2 E4 80 E4 C4 70 8A D8 24 0F 0A C4 B1 03 D2
[UG5] 238: E3 80 E3 80 0A C3 A2 09 00 C2 04 0A B4 0F CD 10
[TR5] 239: B4 03 CD 10 8B EC 8A 5E 02 80 FB 28 75 20 80 FC
[WG5] 240: 28 74 14 3C 06 74 04 B0 01 EB 02 B0 05 FE CB 88
[3K5] 241: 1E 0B 00 B4 00 CD 10 B4 01 CD 10 C2 02 00 80 FB
[2T5] 242: 50 75 F4 80 FC 50 74 EF 3C 04 73 04 B0 03 EB DD
[NSS] 243: B0 06 EB D9 8B EC 8A 46 04 FE C8 A2 0C 00 8A 46
[WT5] 244: 02 FE C8 A2 0D 00 33 DB B8 DE 1E 8C D0 8E D8 AC B4
[1T5] 245: 04 00 8B F4 83 C6 08 8B DE 1E 8C D0 8E D8 AC B4
[RE5] 246: 00 8B D6 03 F0 88 24 80 7F 05 3A 75 18 8B 47 01
[5R5] 247: 25 DF DF 8B 4F 03 81 E1 DF DF 3D 53 43 75 0E 81
[3P5] 248: F9 52 4E 74 03 E9 C2 06 B0 FF E9 CD 00 3D 4B 59
[4R5] 249: 75 08 81 F9 42 44 74 F0 EB EB 3D 43 47 75 E6 B6
[VT5] 250: 03 81 F9 4D 11 74 08 FE CE 81 F9 4D 12 75 D6 B2
[HP5] 251: FB B0 83 EE 8D 77 06 52 33 C9 AC 2C 30 72 12 3C
[RT5] 252: 0A 73 0E B4 00 8B F8 B0 0A F7 E1 03 C7 8B C8 EB
[3N5] 253: E9 23 C9 75 03 B9 2C 01 B8 00 C2 BA 01 00 F7 F1
[CK5] 254: 5A B2 F8 EE 42 8A C4 EE 42 42 B0 03 EE 42 B0 0B
[ZK5] 255: EE B2 F8 0E 01 EE 4A EE 1E 33 C0 8E D8 B1 FE 80
[7H5] 256: FE 03 75 1F 2E A3 1D 01 B8 21 01 87 06 30 00 2E
[RD5] 257: A3 15 01 8C 8B 87 06 32 00 2E A3 17 01 E4 21 24
[EP5] 258: EF EB 1F FE C9 2E A3 1F 01 B8 51 01 87 06 2C 00
[HH5] 259: 2E A3 19 01 8C 8B 87 06 2E 00 2E A3 1B 01 E4 21
[5S5] 260: 24 F7 E6 21 1F EC 8A C1 EB 1F 8A 47 FE 3C 01 74
[CK5] 261: 12 3C 03 75 02 B0 01 B4 3D CD 21 73 0C 80 7F FE
[ET5] 262: 00 74 64 B1 20 B4 3C CD 21 8A 67 FE 8B 5F FC 1F
[1P5] 263: 88 07 C6 47 04 00 C6 47 10 1A 3C FD 73 46 80 FC
[1N5] 264: 03 75 0D 8A D8 33 C9 33 D2 B8 02 42 CD 21 EB 2C
[MK5] 265: 80 FC 00 75 0B C6 47 10 00 BA 39 03 FF D2 EB 1C
[7L5] 266: 80 FC 02 75 15 53 8A 1F 33 C9 33 D2 B8 02 42 CD
[FF5] 267: 21 5B 72 08 0B C2 74 04 C6 47 10 00 8B EC 8A 46
[RL5] 268: 02 88 47 1A C2 06 01 8B 5F FC 1F C6 07 00 C6 47
[YN5] 269: 04 00 C6 47 10 1A C2 06 01 8B EC 8B 5E 02 BA 77
[XP5] 270: 08 FF D2 C2 02 00 8B EC 8B 5E 02 80 3F FF 75 14
[DM5] 271: B4 03 CD 10 A0 00 00 2C 17 3A D0 72 0A B0 0D BA
[VR5] 272: A3 01 FF E2 8A 57 04 8A C2 B4 00 B9 0E 00 F6 F1
[4R5] 273: 2A CC B0 20 51 BA A3 01 FF D2 59 E2 F5 C3 58 8B
[BL5] 274: EC 8E 46 00 89 46 00 BF 01 00 8B 5E 02 C6 0E 0E
[6M5] 275: 00 00 57 BA 39 03 FF D2 5F 3C 00 74 F5 3C 1A 74
[TR5] 276: 1F 3C 0D 74 09 81 FF 00 01 73 E7 AA EB E4 80 3F
[6P5] 277: FF 74 0D 80 7F 10 0A 75 07 57 BA 39 03 FF D2 5F
[ST5] 278: 8D 45 FF 26 A2 00 00 BA 39 05 FF E2 33 DB B0 3F
[8U5] 279: BA A3 01 FF D2 00 20 BA A3 01 FF E2 81 EC FE 00
[6V5] 280: 8B EC FF B6 FE 00 8C D0 8E C0 8D 7E 01 8B 9E 02
[ZH5] 281: 01 80 26 0E 00 7F E8 54 00 74 49 3C 20 74 F7 3C
[RL5] 282: 22 75 21 E8 47 00 74 3C 3C 22 74 0D 57 2B FD 81
[ES5] 283: FF 00 01 5F 73 ED AA EB EA 8E 31 00 74 26 3C 2C
[UN5] 284: 75 F7 EB 20 3C 2D AA 10 57 2B FD 81 FF 00 01 5F
[BSS] 285: 73 01 AA E8 17 00 75 EC 4F 3B FD 74 06 26 80 3D
[TNS] 286: 20 74 F5 47 8D 45 FF 2B C5 88 46 00 C3 57 BA 39
```

```
[RM5] 287: 03 FF D2 5F 3C 1A 74 15 3C 0D 75 11 80 7F 10 0A
[PUS] 288: 75 09 57 BA 39 03 FF D2 5F B0 0D 3C 0D C3 8B EC
[YG5] 289: 8A 46 02 B4 00 89 86 00 01 C2 FE 00 BB 04 00 BA
[YV5] 290: 77 08 FF D2 4B 75 F8 C3 8B EC 8B 5E 02 B0 FF 38
[2F5] 291: 07 74 06 80 7F 10 1A 74 02 F6 D0 98 89 46 02 C3
[VV5] 292: 81 EC FE 00 8B EC B4 0B CD 21 B4 FF 3C 00 74 08
[BK5] 293: B4 08 CD 21 8A E0 B0 01 89 46 00 80 FC 00 75 0A
[GS5] 294: B4 08 CD 21 88 46 02 FE 46 00 FF A6 FE 00 BA 54
[3R5] 295: 05 FF D2 33 D2 A8 01 74 01 4A 89 96 FE 01 C2 FE
[LM5] 296: 01 BA 54 05 FF D2 33 D2 A8 02 74 01 4A 89 96 FE
[PN5] 297: 01 C2 FE 01 BA 54 05 FF D2 33 D2 A8 04 74 01 4A
[9P5] 298: 89 96 FE 01 C2 FE 01 BA 54 05 FF D2 33 D2 A8 03
[WM5] 299: 74 01 4A 89 96 FE 01 C2 FE 01 BA 54 05 FF D2 33
[KM5] 300: D2 A8 05 74 01 4A 89 96 FE 01 C2 FE 01 BA 54 05
[UN5] 301: FF D2 33 D2 A8 01 75 01 4A 89 96 FE 01 C2 FE 01
[TM5] 302: 5E 5B 59 81 EC 00 01 8B EC 88 4E 00 E3 16 C6 06
[8M5] 303: 0E 00 FF 45 51 BA 39 03 FF D2 59 3C 1A 75 05 88
[MW5] 304: 46 00 E2 EF 8B EC 28 4E 00 FF E6 8B EC 8B 5E 02
[VX5] 305: BA 39 05 FF E2 8B EC 8A 8E 02 01 8A 6E 02 80 FD
[EM5] 306: 00 74 2B 8B 9E 02 02 8A C3 FE C8 02 C5 72 1F 3A
[VP5] 307: C1 77 1B 8B F3 BF 01 00 8A D5 46 47 8A 82 01 01
[3P5] 308: 3A 43 01 75 06 FE CA 75 F1 8B 05 43 EB D9 33 DB
[PT5] 309: 89 9E 02 02 C2 00 02 8B F4 8C DA 8C D0 8E D8 AD
[Z25] 310: AD 8E C0 33 FF FC AC AA C8 B5 00 F3 A4 8E D8
[2L5] 311: C2 02 01 8B EC 8B 46 02 87 46 04 83 C4 8F E0
[SU5] 312: 8B FA 83 C6 02 8C DA 8C D0 8E D8 8C 08 B9 9C 00
[4U5] 313: 01 8A 0C B5 00 03 F1 8D 7C 02 41 FD F3 A4 FC 8E
[ET5] 314: DA 83 C4 04 FF E3 8B EC 8E 06 19 00 8B 7E 04 8A
[WK5] 315: 46 02 AA C2 04 00 8B EC 8E 06 19 00 8B 7E 02 86
[1M5] 316: 8A 05 B4 00 89 46 02 C3 8B EC 8B 46 02 33 C0 80
[XP5] 317: F9 01 72 02 8A C5 89 86 00 01 C2 FE 00 8B EC 8A
[DH5] 318: 46 04 8A 66 02 3A E0 73 02 8A C4 88 46 04 C2 02
[6T5] 319: 00 8B FC 83 C7 04 36 8A 4D FE 36 8A 1D 2A D9 76
[ZP5] 320: 16 B7 00 8D 71 01 8C DA 8C D0 8E D8 8E C0 B5 00
[7U5] 321: 8A C1 AA F3 A4 8E DA C2 02 00 8B FC 83 C7 06 8C
[4Y5] 322: DA 8C D0 8E D8 8E C0 8A 5D FE 80 FB 00 75 02 B3
[9M5] 323: 01 B7 00 8D 31 8A 0D B5 00 FE CB 2A CB 73 02 B1
[C75] 324: 00 8A 45 FC 3A C8 76 02 8A C8 8A C1 AA F3 A4 8E
[LS5] 325: DA C2 04 00 8B FC 83 C7 04 8C D0 8E C0 26 8A 8D
[2M5] 326: FE 00 B5 00 26 8B 45 FE 3C 00 75 02 B1 00 8A C1
[G75] 327: AA 8A C4 F3 AA C2 02 00 8B EC 8B 46 04 8E C0 8C
[6M5] 328: DB 2B C3 B1 04 D3 E0 40 8B 5E 02 89 47 01 26 A0
[XH5] 329: 00 00 88 07 89 5E 04 C2 02 00 8B EC 8C 8B 46 04
[2W5] 330: 02 8C DB 8E C3 53 50 CB 8B F4 83 C6 02 8C DA 8C
[SN5] 331: D0 8E D8 8A 8C 00 01 B5 00 AC 3A C1 73 02 8A C8
[EE5] 332: 8E 84 03 01 8A 9C 01 01 B7 00 80 FB 00 77 01 43
[1L5] 333: 8B FB 26 A0 00 00 2A C3 72 0A FE C0 3A C8 76 02
[KX5] 334: 8A C8 F3 A4 8E DA C2 06 01 81 EC FE 00 8B EC B6
[HH5] 335: 2C D2 21 C6 46 00 8B 8A C5 E8 1F 00 89 46 01 C4
[XP5] 336: 46 03 3A 8A C1 E8 13 00 89 46 04 C6 46 06 3A 8A
[XK5] 337: C6 E8 07 00 89 46 07 FF A6 FE 00 B4 00 30 A F6
[8K5] 338: F3 05 30 30 C3 8B EC 8A 46 02 8B 76 04 8B 5E 06
[UV5] 339: 88 04 D1 E3 89 77 1C AC 8A D0 AD B4 00 8B C8 AD
[SR5] 340: 8E C0 26 88 0E 00 BF 01 00 B0 20 F3 AA FE CA
[Y55] 341: 75 E8 C2 06 00 8B EC 8B 5E 06 D1 E3 8B 77 1C D1
[ZM5] 342: EB 8A 47 1A B4 00 50 8B 56 02 8B 4E 04 80 C9 80
[3V5] 343: 80 FD 98 73 08 D0 E9 D1 DA FE C5 EB F3 83 EA 01
[3R5] 344: 80 D9 00 F6 E1 8B C8 58 50 F7 E2 03 D1 80 80 8B
[3L5] 345: FB 8A 1F B8 00 42 CD 21 59 57 03 C1 83 D2 00 8E
[LM5] 346: C0 8B EA AC B4 00 8B F8 AD 8B C8 1E AD 8E D8 88
[F15] 347: 0E 00 00 BA 01 00 B4 3F CD 21 1F 72 24 3B C1 75
[FK5] 348: 20 4F 75 E4 33 C9 33 D2 B8 02 42 CD 21 72 12 3B
[RM5] 349: EA 75 06 8C C6 3B F0 74 08 5B C6 47 10 00 C2 06
[9N5] 350: 00 5B C6 47 10 1A C2 06 00 8B F4 83 C6 02 8C DA
[4H5] 351: 8C D0 8E D8 8E 84 00 01 BF 01 00 26 8A 0E 00 7F
[RT5] 352: B0 03 00 AC 3A C1 73 06 8A D9 2A D8 8A C8 F3
[XR5] 353: A4 8A CB B0 20 F3 AA 8E DA C2 02 01 B0 F4 83 C6
[FS5] 354: 02 8C DA 8C D0 8E D8 8E 84 00 01 BF 01 00 26 8A
[LL5] 355: 1E 00 00 B5 00 B1 00 00 C3 A3 C3 73 06 8A CB 2A C8
[3U5] 356: BA D8 B0 20 F3 AA 8A CB F3 A4 8E DA C2 01 8B
[SR5] 357: EC 8B 5E 06 D1 E3 8B 77 1C D1 EB 8A 47 1A B4 00
[HY5] 358: 50 8B 56 02 8B 4E 04 80 C9 80 80 FD 98 73 08 D0
[NY5] 359: E9 D1 DA FE C5 EB F3 83 EA 01 80 D9 00 F6 E1 8B
[RR5] 360: C8 58 F7 E2 03 D1 8B D0 8A 1F B8 00 42 CD 21 AC
[PW5] 361: B4 00 8B F8 AD 8B C8 1E AD 8E D8 BA 01 00 B4 40
[OM5] 362: CD 21 1F 72 07 3B C1 75 03 4F 75 E8 C2 06 00 8B
[BM5] 363: EC 8B 7E 00 8B 46 02 B3 00 23 C0 74 15 BB 90 7F
[FK5] 364: 79 04 F7 D8 B7 FF 23 C0 78 06 D1 E0 FE CB EB F6
[XK5] 365: 22 E7 C6 46 00 89 46 01 88 5E 03 FF E7 81 EC
[NR5] 366: FA 00 8B FC B2 00 36 8A B5 FC 00 36 8B 5F FD 00
[OP5] 367: 36 8A 9D FF 00 8C D0 8E C0 47 B0 20 23 F6 79 02
[FU5] 368: B0 2D AA 80 FB 00 75 14 B0 30 AA 8B EC 8D 45 FF
[GL5] 369: 2B C5 88 46 00 B0 20 AA FF A6 FA 00 81 CE 00 80
[FP5] 370: 33 C9 80 FB 81 72 03 E9 D1 00 E8 7A 00 FE C9 80
[LB5] 371: FB 80 72 F6 81 C2 80 00 83 D6 00 73 06 D1 DE D1
[GS5] 372: DA FE C3 80 FB 81 73 05 E8 5C 00 FE C9 E8 86 00
[OP5] 373: 2A E9 80 FD 07 77 33 8B EC 8B 46 02 86 C4 89 46
[5K5] 374: 02 83 C5 03 3B FD 75 01 47 8B F7 F6 D1 B5 00 03
[4M5] 375: F9 57 4E 4F 36 8A 04 36 88 05 3B F5 77 F4 E3 07
[7R5] 376: 4F 36 C6 05 30 E2 F9 5F EB 81 B0 45 AA B0 2B 22
[HS5] 377: C9 79 04 F6 D9 B0 2D AA 8A C1 B4 00 B1 0A F6 F1
[WY5] 378: D0 30 30 AB E9 64 FF 8B C2 8B EE D1 ED D1 D8 D1
[ZT5] 379: ED D1 D8 13 D0 13 F5 73 06 D1 DE D1 DA FE C3 80
```

Listing 3. Die assemblierte Laufzeitbibliothek »sbasic.bin« als Hexdump


```
[0V5] 380: C3 03 C3 B0 00 D1 E2 D1 D6 D0 D0 FE CB 80 FB 80
[EY5] 381: 77 F3 0C 30 AA C3 E8 EA FF B0 2E AA E8 C8 FF E8
[UP5] 382: E1 FF FE C5 80 FD 06 72 F3 26 8A 45 FF 3C 30 75
[0V5] 383: 05 FE CD 4F EB F3 3C 2E 74 F9 C3 E8 4B 00 FE C1
[4H5] 384: 80 FB 81 73 F6 FE C9 81 C2 80 00 83 D6 00 73 06
[826] 385: D1 DE D1 DA FE C3 E8 8E FF E8 BA FF 80 F9 07 72
[4S5] 386: 03 E9 66 FF 8B EC 83 C5 03 FE C9 78 0B 8B 46 00
[LL5] 387: 86 C4 89 46 00 45 EB F1 3B FD 73 02 8B FD 36 80
[AT5] 388: 7D FF 2E 75 01 4F E9 C2 FE 52 BD 0A 00 8B C6 33
[4V5] 389: D2 F7 F5 8B F0 58 F7 F5 92 B4 1A F6 E4 D0 E0 D1
[WS5] 390: D2 D1 D6 FE CB 23 F6 79 F4 C3 8B EC 80 76 04 80
[5U5] 391: C3 BA 86 05 FF E2 83 EC 02 8B EC 8B 46 06 87 46
[7C5] 392: 04 87 46 02 89 46 00 8B 46 08 B3 00 23 C0 74 15
[HW5] 393: BB 90 7F 79 04 F7 D8 B7 FF 23 C0 78 06 D1 E0 FE
[YS5] 394: CB EB F6 22 E7 C6 46 06 00 89 46 07 88 5E 09 C3
[LS5] 395: BA 22 06 FF E2 BA BE 06 FF E2 BA 4A 07 FF E2 8B
[NM5] 396: EC 8B 46 08 23 C0 74 2E 50 24 7F 50 FF 76 06 BA
[BX5] 397: A4 07 FF D2 8B EC FF 76 0A FF 76 0A BA 22 06 FF
[VU5] 398: D2 BA D0 08 FF D2 58 5B 59 80 E1 80 0A D9 8B EC
[FN5] 399: 89 46 06 89 5E 08 C2 04 00 8B EC 8B 5E 04 4B 78
[SH5] 400: 15 D1 E3 03 5E 02 8B 56 00 83 EA 03 3B DA 73 06
[SN5] 401: 2E 8B 07 89 46 00 C2 04 00 8B EC 8B 5E 04 4B 78
[LM5] 402: 1B D1 E3 03 5E 02 8B 56 00 83 EA 03 3B DA 73 0C
[4E5] 403: 2E 8B 07 87 46 00 89 46 04 C2 02 00 C2 04 00 BA
[7N5] 404: E3 09 FF E2 8B EC 8B 46 03 8A 6E 05 22 ED 74 16
[8N5] 405: 8A DC 80 CC 80 B1 90 2A CD 72 0B D3 E8 15 00 00
[HD5] 406: 22 DB 79 02 F7 D8 89 46 04 C2 02 00 07 BA 3F 08
[EG5] 407: FF D2 33 C0 80 FA D1 75 01 48 89 46 08 06 C2 86
[DS5] 408: 00 07 BA 3F 08 FF D2 33 C0 80 FA FF 75 01 48 89
[PN5] 409: 46 08 06 C2 06 00 07 BA 3F 08 FF D2 33 C0 80 FA
[NF5] 410: 00 75 01 48 89 46 08 06 C2 06 00 07 BA 3F 08 FF
[ZE5] 411: D2 33 C0 80 FA FF 74 01 48 89 46 08 06 C2 06 00
[BG5] 412: 07 BA 3F 08 FF D2 33 C0 80 FA 01 74 01 48 89 46
[AM5] 413: 08 06 C2 06 00 07 BA 3F 08 FF D2 33 C0 80 FA 00
[WG5] 414: 74 01 48 89 46 08 06 C2 06 00 8B EC 8B 46 02 87
[AM5] 415: 46 04 87 46 06 83 C4 04 FF E0 8B EC 33 D2 8B 46
[5F5] 416: 04 80 FC 00 74 07 42 22 C0 79 02 F7 DA 89 56 04
[6N5] 417: C2 02 00 8B EC 8A 4E 02 22 C9 74 02 FE C9 8B 5E
[8L5] 418: 04 80 3F FF 74 25 38 4F 04 76 10 51 B0 0D BA A3
[GN5] 419: 01 FF D2 B0 0A BA A3 01 FF D2 59 38 4F 04 73 42
[7T5] 420: 51 B0 20 BA A3 01 FF D2 59 38 4F 04 73 42
[5N5] 421: CC 73 FC 02 CC 51 B4 03 CD 10 59 51 3A D1 76 09
[RU5] 422: 33 DB B0 0D BA A3 01 FF D2 B4 0F CD 10 B4 03 CD
[EM5] 423: 10 59 3A D1 73 0C 51 33 DB B0 20 BA A3 01 FF D2
[8S5] 424: EB E7 C2 02 00 8B EC 8B 4E 02 83 F9 00 76 0E 8B
[XH5] 425: 5E 04 51 B0 20 BA A3 01 FF D2 59 E2 F5 C2 02 00
[4T5] 426: 8B EC BA 0F CD 10 B4 03 CD 10 8A D6 B6 00 42 87
[2U5] 427: 56 00 FF E2 8B EC BA 0F CD 10 B4 03 CD 10 B6 00
[LS5] 428: 42 87 56 00 FF E2 8B EC 8A 4E 02 8A 6E 04 89 0E
[XE5] 429: 24 00 83 7E 06 00 75 03 B9 00 1F B4 01 CD 10 C2
[WM5] 430: 06 00 8B EC BA 0F CD 10 B4 03 CD 10 81 E9 00 1F
[ER5] 431: 87 4E 00 FF E1 8B EC A0 25 00 B4 00 87 46 00 FF
[BR5] 432: E0 8B EC A0 24 00 B4 00 87 46 00 FF E0 8B EC A0
[DH5] 433: 09 00 8A D8 25 00 0B 01 04 D2 E8 80 E3 10 0A C3
[FM5] 434: 87 46 00 FF E0 8B EC A0 09 00 B1 04 D2 E8 25 07
[6P5] 435: 00 87 46 00 FF E0 1E 8C D0 8E D8 8B F4 83 C6 04
[KR5] 436: AC 8A C8 B5 00 33 DB 33 D2 33 ED E3 4D E8 60 01
[OR5] 437: 74 48 3C 2B 74 F7 3C D2 75 04 F7 D5 EB EF 3C 2E
[LM5] 438: 75 6A E8 4B 01 74 33 3C 68 74 04 3C 48 75 2B E8
[9G5] 439: 3E 01 74 26 C2 30 72 22 3C 0A 72 12 2C 07 3C 0A
[SB5] 440: 72 18 3C 10 72 08 2C 20 72 10 3C 10 73 0C D1 E2
[KN5] 441: D1 E2 D1 E2 D1 E2 02 D0 EB D5 23 D2 74 1C B3 9B
[ZY5] 442: 79 04 F7 DA F7 D2 22 F6 78 06 D1 E2 FE CB EB F6
[RL5] 443: 80 E6 7F 23 ED 74 03 80 CE 80 8B EC 88 BE 00 FF
[KH5] 444: 89 96 01 01 88 9E 03 01 1F C2 FC 00 55 33 FF 3C
[WM5] 445: 2E 75 53 23 FF 75 06 4F E8 D5 00 75 F2 23 FF 74
[EE5] 446: 01 47 23 FF 74 2A 57 53 52 78 12 8B 20 84 50 33
[CV5] 447: C0 50 B8 22 06 FF D2 5A 5B 5F 4F EB 5E B8 4C 7D
[VU5] 448: 50 B8 CD CC 50 BA 22 06 FF D2 5A 5B 5F 4F EB D2
[PL5] 449: 58 23 C0 74 03 80 F3 80 8B EC 89 96 00 01 89 9E
[SG5] 450: 02 01 1F C2 FC 00 3C 65 74 04 3C 45 75 42 23 FF
[YP5] 451: 74 01 47 33 ED 8E C5 E8 76 00 74 A6 3C 2B 74 F7
[HT5] 452: 3C 2D 75 08 8C C0 F7 D0 8E C0 EB EB 2C 30 72 10
[SS5] 453: 3C 0A 73 0C 95 B4 0A F6 E4 03 E8 E8 52 00 75 EC
[2V5] 454: 8B C5 98 8C C5 23 ED 79 02 F7 D8 03 F8 E9 72 FF
[WM5] 455: B4 00 2C 30 73 03 E9 64 FF 74 0E 3C 0A 73 F7 B4
[FN5] 456: 88 FE CC D0 E0 79 FA 24 7F 51 56 57 50 33 C0 50
[RG5] 457: B8 20 84 50 33 C0 50 53 52 BA 22 06 FF D2 BA 86
[HT5] 458: 05 FF D2 5A 5B 5F 5E 59 23 FF 74 01 4F E9 28 FF
[3R5] 459: 23 C9 74 05 AC 3C 20 E1 F7 C3 8B EC 8B 46 05 8A
[8S5] 460: 6E 07 22 ED 74 16 8A DC 80 CC 80 B1 90 2A CD 72
[UN5] 461: 0B D3 E8 15 00 00 22 DB 79 02 F7 D8 89 46 06 8B
[7C5] 462: 46 02 89 46 04 C2 02 00 8B EC 33 D2 80 7E 02 00
[SR5] 463: 74 01 4A 89 96 00 01 C2 FE 00 81 EC FC 00 8B EC
[PL5] 464: 8B 86 FE 00 89 46 01 C6 46 00 02 FF A6 FC 00 81
[TS5] 465: EC FA 00 8B EC 8B 86 FC 00 89 46 01 8B 86 FE 00
[8N5] 466: 89 46 03 C6 46 00 04 FA 00 8B EC 8B 46 04
[KK5] 467: 22 C0 79 0A 8B 5E 02 89 1E 26 00 A3 28 00 FF 36
[TM5] 468: 28 00 FF 36 26 00 22 E4 74 4B B8 A7 98 50 B8 E0
[UN5] 469: 18 50 BA 22 06 FF D2 B8 28 68 50 B8 46 B1 50 BA
[FP5] 470: 86 05 FF D2 8B 46 06 23 80 5E 08 34 AA 80 F3 55 86
[CL5] 471: C3 80 F4 C3 B7 00 06 C0 75 04 22 DB 74 11 B7 80
[TT5] 472: 22 DB 78 08 D1 E0 D0 D3 FE CF EB F4 80 E3 7F 89
[UK5] 473: 46 06 89 5E 08 D0 D0 D3 FE CF EB F4 80 E3 7F 89
[5N5] 474: 46 08 A3 28 00 C3 8B EC B4 0F CD 10 B4 03 CD 10
```

```
[EF5] 475: 8B F2 8A 56 04 8A 76 06 81 EA 01 01 B4 02 CD 10
[1K5] 476: B4 08 CD 10 8B 4E 02 23 C9 74 02 8A C4 B4 00 89
[PL5] 477: 46 06 8B D6 B4 02 CD 10 C2 04 00 81 EC FE 00 8B
[GM5] 478: EC B4 2A CD 21 8A C6 E8 38 00 89 46 01 C6 46 03
[PL5] 479: 2D 8A C2 E8 2C 00 89 46 04 C6 46 06 2D B8 31 39
[CD5] 480: 81 E9 6C 07 80 F9 64 72 06 B8 32 30 80 E9 64 89
[NM5] 481: 46 07 8A C1 E8 0B 00 89 46 09 C6 46 00 0A FF A6
[HM5] 482: FE 00 B4 00 B3 0A F6 F3 05 30 30 C3 83 EC 02 B4
[RU5] 483: 00 CD 1A 8A DA 8A D6 8A F1 8A CD B5 00 22 DB 75
[7H5] 484: 08 23 D2 75 04 23 C9 74 13 B5 A0 22 C9 78 0A D0
[XV5] 485: E3 D1 D2 D0 D1 FE CD EB F2 80 E1 7F 51 52 B8 60
[AT5] 486: 7C 50 B8 7A F9 50 BA 22 06 FF D2 8F 46 0A 58 87
[UK5] 487: 46 0C FF E0 8B EC 33 C0 80 7E 02 02 75 03 8B 46
[9K5] 488: 03 89 86 00 01 C2 FE 00 8B EC 33 C0 33 DB 80 7E
[FM5] 489: 02 04 75 06 8B 46 03 8B 5E 05 89 86 FE 00 89 9E
[XT5] 490: 00 01 C2 FC 00 8B EC 80 7E 05 98 73 30 B8 FF FF
[GM5] 491: B3 FF B1 90 2A 4E 05 73 07 80 C1 08 D2 E8 E0 BA 17
[3L5] 492: B0 00 80 E9 08 73 07 80 C1 08 D2 E8 E0 BA 17
[GE5] 493: 80 F9 08 73 09 D2 E3 21 46 02 20 5E 04 C3 89 46
[7T5] 494: 02 89 46 04 C3 81 EC FC 00 8B EC 8B 86 FE 00 8B
[VN5] 495: 10 00 33 C9 33 D2 F7 F3 52 23 C0 E0 F7 F7 D9 88
[RB5] 496: 4E 00 BF 01 00 58 04 30 3C 39 76 02 04 07 88 03
[4U5] 497: 47 E2 F2 FF A6 FC 00 8B EC 8B 56 02 EC B4 00 89
[FN5] 498: 46 02 C3 8B EC 83 C5 02 8B 9E 00 01 8A 4E 00 B5
[YF5] 499: 00 E3 11 45 8A 46 03 C0 3C 20 74 07 51 BA A3 01 FF
[6R5] 500: D2 59 E2 EF C2 00 01 8B EC 83 C5 02 8B 9E 00 01
[NR5] 501: B0 22 BA A3 01 FF D2 8A 4E 00 B5 00 83 0D 45 8A
[UP5] 502: 46 00 51 BA A3 01 FF D2 59 E2 F3 B0 22 BA A3 01
[5U5] 503: FF D2 C2 00 01 8B EC 8B 5E 02 B6 2C BA A3 01 FF
[ES5] 504: E2 8B EC 8A 5E 02 8A 4E 04 8B 56 06 EC 32 C3 22
[MR5] 505: C1 74 F9 C2 06 00 8B EC 83 C5 03 8A 4E FF B5 00
[6P5] 506: E8 15 00 8A FE E8 10 00 8A DE 08 8A 4E FF B5 00
[LG5] 507: CB BA 2D CD 21 C2 00 01 B6 00 E8 14 00 2C 30 72
[HP5] 508: 19 3C 0A 73 15 50 B0 0A F6 E6 8A F0 58 02 F0 EB
[6L5] 509: E9 B0 00 E3 05 8A 46 00 45 49 C3 8B EC 8B 76 02
[NK5] 510: 8B 7E 04 8B 04 87 05 89 04 C2 04 00 8B EC 8C D8
[4L5] 511: 8E C0 8B 76 02 8B 7E 04 26 8B 05 A5 89 4E FE 26
[WS5] 512: 8B 05 A5 89 44 FE C2 04 00 8B EC 8C DA 8E 5E 02
[6G5] 513: 8E 46 04 8A 0E 00 26 8A 2E 00 80 3A CD 73 02
[7R5] 514: 8A CD B5 00 41 33 F6 33 FF 26 8A 05 A4 88 44 FF
[5S5] 515: E2 F7 8E DA C2 04 00 8B EC B8 11 95 50 B8 F0 A6
[2V5] 516: 50 FF 76 08 FF 76 06 BA BE 06 FF D2 58 5B 22 FF
[XS5] 517: 74 12 80 CB 80 8A C4 8A E3 80 FF 90 73 06 D1 E8
[KX5] 518: FE C7 EB F5 8B 5E 0E 22 FF 74 13 8B 4E 0C 80 CB
[WW5] 519: 80 80 FF 90 73 08 D0 EB D1 D9 FE C7 EB F3 22 DB
[7L5] 520: 75 02 E3 2C 50 B0 B6 E6 43 B0 14 E6 43 00 E6
[WH5] 521: 40 58 E6 42 8A C4 E6 42 E4 61 0C 03 E6 61 E4 40
[XV5] 522: 22 C0 79 FA E4 40 22 C0 78 FA E2 F2 FE CB 79 EE
[XM5] 523: E4 61 24 FD E6 61 B0 36 E6 43 B0 00 E6 40 E6 40
[NR5] 524: C2 08 00 81 EC FE 00 8B EC C6 46 00 00 FF A6 FE
[WL5] 525: 00 8B 0E 0F 00 B4 01 CD 10 83 EC 02 8B EC 8B 76
[PC5] 526: 02 8A 46 04 B4 0D 89 46 01 C6 46 03 00 22 C0 74
[LF5] 527: 1E 04 03 B4 20 89 46 01 C7 46 03 2F 43 B4 00 8B
[NH5] 528: F8 81 FF 00 01 76 03 BF 00 01 C7 43 02 D0 00 1E
[GL5] 529: 56 16 45 55 8E 1E 2A 00 8E 1E 2C 00 1E 33 F6 E8
[6L5] 530: 08 00 43 4F 4D 53 50 45 43 1D 5B 8B D6 AC 22 C0
[TS5] 531: 74 1B 8B FB B9 08 00 24 DF 47 2E 3A 45 FF AC E1
[SU5] 532: F6 74 07 AC 22 C0 75 FB EB E1 83 C2 08 8C D0 8E
[AM5] 533: C0 8B DC E8 00 00 5F 2E 8C 55 12 90 2E 89 65 17
[5R5] 534: 90 B8 00 4B CD 21 FA B8 00 00 8E D0 BC 00 00 FB
[8M5] 535: 8B EC 8E 5E 08 8B 6E 06 81 C4 0E 01 B4 0F CD 10
[2L5] 536: B4 03 CD 10 80 FE 18 72 B0 8A 0A 0E CD 10 B6 17
[KP5] 537: B4 02 CD 10 8B 0E 24 00 B4 01 CD 10 FF E5 8B EC
[7L5] 538: B4 0F CD 10 B4 03 CD 10 8A 46 02 B4 05 CD 10 B4
[9K5] 539: 0F CD 10 B4 02 CD 10 C2 06 00 8C DA 83 C2 03 83
[YM5] 540: EB 03 74 0F 33 C0 8E C2 33 FF B9 08 00 F3 AB 42
[7W5] 541: 4B 75 F3 BB 0A 00 BA 77 08 FF D2 48 75 F8 C3 1E
[4X5] 542: 8C D0 8E D8 8B FC 83 C7 05 8A 5D FF 8B FF 75
[UP5] 543: 02 FE CB B7 00 88 39 8B D7 B4 39 CD 21 1F C2 00
[XV5] 544: 01 1E 8C D0 8E D8 8B FC 83 C7 05 8A 5D FF 8B FF
[DV5] 545: FF 75 02 FE CB B7 00 88 39 8B D7 B4 3A CD 21 1F
[7U5] 546: C2 00 01 1E 8C D0 8E D8 8B FC 83 C7 05 8A 5D FF
[2U5] 547: 80 FB FF 75 02 FE CB B7 00 88 39 8B D7 B4 3B CD
[FG5] 548: 21 1F C2 00 01 5A 8F 06 26 00 8F 06 28 00 FF E2
[2V5] 549: B8 07 0E CD 10 C3 81 EC FC 00 8B FC 8C D0 8E C0
[NR5] 550: 36 8A 85 FE 00 B4 00 8B C8 36 8B 95 FC 00 AA B0
[8W5] 551: 20 F3 AA FF E2 8B EC 8E 06 19 00 1E 8C D0 8E D8
[VK5] 552: 8D 76 06 8A 0C B5 00 80 F9 FA 72 02 B1 FA 03 F1
[CG5] 553: 8D 7C 01 80 F9 04 76 02 B1 04 E3 08 80 3C 2E 74
[XK5] 554: 0F 4E E2 F8 C7 05 2E 42 C7 05 02 41 53 83 C7 04
[HL5] 555: C6 05 00 8D 56 07 B1 20 B4 3C CD 21 72 33 8B D8
[AF5] 556: 8C C0 87 46 04 89 46 06 B4 FD 87 46 02 89 46 08
[WK5] 557: B9 07 00 8D 56 03 B4 40 CD 21 72 11 3B C1 75 0D
[K55] 558: 8E 5E 04 8B 56 06 8B 4E 08 B4 40 CD 21 B4 3E CD
[YP5] 559: 21 1F C2 04 01 8B EC 8E 06 19 00 1E 8C D0 8E D8
[TK5] 560: 8D 76 04 8A 0C B5 00 80 F9 FA 72 02 B1 FA 03 F1
[CG5] 561: 8D 7C 01 80 F9 04 76 02 B1 04 E3 08 80 3C 2E 74
[XK5] 562: 0F 4E E2 F8 C7 05 2E 42 C7 05 02 41 53 83 C7 04
[PM5] 563: C6 05 00 8D 56 05 8B 00 3D CD 21 72 35 8B D8 B9
[HG5] 564: 07 00 B4 3F CD 21 72 26 3D 07 00 75 21 8B D8 B9
[TT5] 565: FD 75 1B 8B 56 02 83 FA FF 75 08 8E 5E 06 8B 56
[EY5] 566: 08 EB 04 8C 0E 8B D8 8B 4E 0A B4 3F CD 21 B4 3E
[LR5] 567: CD 21 1F C2 02 01 8B EC 8A 5E 02 B7 00 B4 0B CD
[DD5] 568: 10 C2 02 00 8B EC 8B 56 04 8A 46 02 EE C2 04 00
[FP5] 569: B0 B6 E6 43 B0 14 E6 43 B0 00 E6 40 8B EC 8A 4E
[UD5] 570: 02 83 C5 03 E8 48 02 22 C0 75 0D B0 36 E6 43 B0
```



```
[6G5] 571: 00 E6 40 E6 40 C2 00 01 3C 41 72 2E 3C 47 77 2A
[9D5] 572: 2C 41 B4 00 E8 07 00 0A 0C 01 03 05 06 08 5B 03
[7H5] 573: D8 2E 8A 07 02 06 2C 00 0A D8 E8 12 02 8A 3E 2D
[7F5] 574: 00 3C 23 75 30 FE C3 E9 87 00 3C 50 75 04 B0 00
[UL5] 575: EB E6 3C 4F 75 11 E8 F6 01 E8 07 02 50 B0 0C F6
[6P5] 576: E2 A2 2C 00 58 EB A0 3C 4E 75 12 E8 E1 01 E8 F2
[GU5] 577: 01 8A DA EB C8 3C 2B 75 52 FE C3 EB 54 3C 4C 75
[2N5] 578: 0D E8 CB 01 E8 DC 01 88 16 2D 00 E9 79 FF 3C 54
[NR5] 579: 75 0D E8 BA 01 E8 CB 01 88 16 2E 00 E9 68 FF 3C
[EK5] 580: 3C 75 12 80 3E 2C 00 0C 72 05 80 2E 2C 00 0C E8
[WK5] 581: 9D 01 E9 52 FF 3C 3E 75 35 80 3E 2C 00 48 73 05
[KN5] 582: 80 06 2C 00 0C E8 87 01 E9 3C FF 3C 2D 75 05 FE
[HK5] 583: CB E8 7B 01 3C 30 72 09 3C 39 77 05 E8 84 01 8A
[SS5] 584: FA B4 00 3C 2E 75 33 FE C4 E8 63 01 EB F5 3C 4D
[BK5] 585: 75 22 E8 5A 01 3C 4E 75 07 C6 06 2F 00 07 EB 14
[RG5] 586: 3C 4C 75 07 C6 06 2F 00 08 EB 09 3C 53 75 05 C6
[AM5] 587: 06 2F 00 06 E8 38 01 E9 ED FE E8 AA 00 01 00 43
[GM5] 588: 47 43 43 7C 3F EC 3B 8F 3E 62 35 63 02 8F 2F E4
[AP5] 589: 2C 5F 2A FE 27 C0 25 A1 23 A1 21 BE 1F F6 1D 48
[XE5] 590: 1C B1 1A 32 19 C8 17 72 16 30 15 FF 13 E0 12 D1
[SL5] 591: 11 D1 10 DF 0F FB 0E 24 0E 59 D0 99 0C E4 0B 39
[KD5] 592: 0B 98 0A 00 0A 70 09 E8 08 68 08 F0 07 7E 07 12
[YD5] 593: 07 AC 06 4C 06 F2 05 9D 05 4C 05 00 05 B8 04 74
[YE5] 594: 04 34 04 F8 03 BF 03 89 03 56 03 26 03 F9 02 CE
[3F5] 595: 02 A6 02 80 02 5C 02 3A 02 1A 02 FC 01 DF 01 C4
[X95] 596: 01 AB 01 93 01 7C 01 67 01 53 01 40 01 2E 01 1D
[YG5] 597: 01 0D 01 FE 00 F0 0E 02 00 D6 00 CA 00 BE 00 B4
[FF5] 598: 00 A9 00 A0 00 97 00 5E 50 8A C3 B4 00 D1 E0 03
[PS5] 599: F0 2E 8B 04 E6 42 8A C4 E6 42 8A DF B7 00 D1 E3
[4P5] 600: D1 E3 D1 E3 D1 E3 58 50 8A C4 B4 00 8B F8 B0 A0
[9N5] 601: F6 26 2F 00 8B F7 4E 78 0B BA 03 00 F7 E2 D1 EA
[TR5] 602: D1 D8 EB F2 8A 16 2E 00 B6 00 F7 E2 F3 8B F0
[OU5] 603: B8 00 05 4F 78 0B BA 03 00 F7 E2 D1 EA D1 DB EB
[HR5] 604: F2 8A 16 2E 00 B6 00 F7 E2 F3 8B F8 E4 61 0C
[3N5] 605: 03 E6 61 E4 40 22 C0 78 FA E4 40 22 C0 79 FA 4E
[4T5] 606: 75 06 E4 61 24 FD E6 61 4F 75 E8 58 E9 B8 FD B0
[VG5] 607: 00 E3 0F 8A 46 00 3C 61 72 06 3C 7A 77 02 24 DF
[GS5] 608: 45 49 C3 BA 00 0A 2C 30 86 C2 F6 E6 02 D0 E8 DE
[WN5] 609: FF 3C 30 72 04 3C 39 76 ED C3 8B EC F6 46 03 80
[9S5] 610: 79 03 F7 5E 02 C3 8B EC 80 66 04 7F C3 1E 8C D0
[RZ5] 611: 8E D8 8B FC 83 C7 05 8A 5D FF 80 FB FF 75 02 FE
[NN5] 612: CB B7 00 88 39 8B D7 B4 41 CD 21 1F C2 00 01 8B
[ZR5] 613: EC 83 C5 03 8A 5E FF 22 DB 75 0B C7 46 00 2A 2E
[GR5] 614: C6 46 02 2A B3 03 80 FB FF 75 02 FE CB B7 00 03
[3F5] 615: DD B2 00 36 88 17 8A 46 00 45 22 C0 74 12 3C 3A
[RL5] 616: 75 10 24 DF 2C 40 72 08 3C 1A 77 04 8A D0 EB 02
[SL5] 617: B2 00 22 D2 75 08 B4 19 CD 21 8A D0 FE C2 52 33
[HV5] 618: DB 8A C2 04 40 BA A3 01 FF D2 B0 3A BA A3 01 FF
[VV5] 619: D2 B0 5C BA A3 01 FF D2 5A 52 83 EC 40 8B F4 1E
[2M5] 620: 8C D0 8E D8 B4 47 CD 21 1F 36 8A 04 46 22 C0 74
[VP5] 621: 07 BA A3 01 FF D2 EB F1 83 C4 40 B0 D0 BA A3 01
[CY5] 622: FF D2 B4 2F CD 21 8D 77 1E 33 DB 1E 8C D0 8E D8
[9N5] 623: 8B D4 83 C2 07 B1 31 B4 4E CD 21 1F 73 03 E9 93
[6N5] 624: 00 3D ED 8B FE 26 8A 03 22 C0 74 1E 3C 2E 74 08
[LY5] 625: BA A3 01 FF D2 45 EB ED B0 2E 83 FD 08 73 F1 B0
[3U5] 626: 20 BA A3 01 FF D2 45 FB EB EE 83 FD 0C 73 BA B0
[YM5] 627: 20 BA A3 01 FF D2 45 EB F1 BD 06 00 26 F6 44 F7
[GR5] 628: 10 74 26 B0 3C BA A3 01 FF D2 B0 44 BA A3 01 FF
[4T5] 629: D2 B0 49 BA A3 01 FF D2 B0 52 BA A3 01 FF D2 B0
[XR5] 630: 3E BA A3 01 FF D2 BD 01 00 B0 20 BA A3 01 FF D2
[WN5] 631: 4D 75 F6 B4 0F CD 10 B4 03 CD 10 33 DB 80 C2 12
[KR5] 632: 3A 16 0B 00 72 07 B0 0D BA A3 01 FF D2 B4 4F CD
[NN5] 633: 21 E9 68 FF B4 0F CD 10 B4 03 CD 10 33 DB 22 D2
[AU5] 634: 74 07 B0 0D BA A3 01 FF D2 B0 20 BA A3 01 FF D2
[KP5] 635: 5A B4 36 CD 21 F7 E1 F7 E3 BB 0A 00 33 C9 8B F0
```

```
[FN5] 636: 8B C2 33 D2 F7 F3 96 F7 F3 52 96 23 F6 E0 F3 23
[6U5] 637: C0 E0 EF 33 DB F7 D1 58 0C 30 51 BA A3 01 FF D2
[7G5] 638: 59 E2 F4 E8 0F 00 20 42 79 74 65 73 20 66 72 65
[3H5] 639: 65 20 0D 0D 00 5E 2E 8A 04 22 C0 74 08 BA A3 01
[PW5] 640: FF D2 46 EB F1 C2 00 01 1E 8C D0 8E D8 8E C0 8B
[DU5] 641: F4 81 C6 05 01 8A 5C FF 80 FB FF 75 02 FE CB B7
[CV5] 642: 00 88 38 8B D6 8D BC 00 FF 8A 5D FF 80 FB FF 75
[NS5] 643: 02 FE CB 88 39 B4 56 CD 21 1F C2 00 02 8B EC 8B
[7S5] 644: 5E 02 23 DB 75 74 53 33 DB 33 ED 33 F6 33 FF B9
[YS5] 645: 7F 00 E4 61 24 FC 0C 01 E6 61 B0 A4 E6 43 B0 00
[9S5] 646: E6 42 B0 94 E6 43 B0 1F E6 42 BA 01 02 EE EC D0
[OP5] 647: E8 83 D3 00 D0 E8 83 D5 00 D0 E8 83 D6 00 D0 E8
[KG5] 648: 83 D7 00 E4 42 24 10 74 FA E4 42 24 10 75 FA E2
[LS5] 649: DD FB B1 02 D3 E3 D3 E5 D3 E6 D3 E7 E8 08 00 00
[SE5] 650: 00 00 00 00 00 00 00 8B C3 5B 2E 89 07 2E 89 6F
[EL5] 651: 02 2E 89 77 04 2E 89 7F 06 5B E8 00 00 5E 83 E3
[7W5] 652: 03 D1 E3 2E 8B 40 E2 8B EC 89 46 02 C3 8B EC BA
[NF5] 653: 01 02 EC 33 D2 8A 66 02 80 E4 06 75 04 A8 10 EB
[8M5] 654: 14 80 FC 04 75 04 A8 20 EB 0B 80 FC 02 75 04 A8
[SL5] 655: 40 EB 02 A8 80 75 01 4A 89 56 02 C3 BA D0 08 FF
[5P5] 656: E2 8B EC 8B 46 04 23 C0 74 2C 50 24 7F 50 FF 76
[BP5] 657: 02 BA 4A 07 FF D2 8B EC 80 7E 03 00 74 03 FE 4E
[PT5] 658: 03 BA D0 08 FF D2 58 5B 59 80 E1 80 0A D9 8B EC
[GU5] 659: 89 46 02 89 5E 04 C3 8B EC 83 C5 03 8A 4E FF B5
[XN5] 660: 00 E8 1E 00 8A FA E8 19 00 8A DA E8 14 00 83 FA
[LP5] 661: 64 73 04 81 C2 6C 07 8B CA 8B D3 B4 2B CD 21 C2
[7F5] 662: 00 01 33 D2 E8 17 00 2C 30 72 1C 3C 0A 73 18 B4
[SM5] 663: 00 50 B8 0A 00 F7 E2 8B D0 58 46 02 0B E6 B0 00
[7M5] 664: E3 05 8A 46 00 45 49 C3 81 EC FC 00 8B EC 8C D0
[5T5] 665: 8E C0 1E BE 80 00 8E 1E 2A 00 CA 8A C8 B5 00 8B
[VH5] 666: 96 FE 00 23 D2 74 32 8D 7E 01 E8 34 00 74 22 3C
[WL5] 667: 20 74 F7 3C 2C 74 1A 3C 61 72 06 3C 7A 77 02 2C
[MN5] 668: 20 AA E8 1C 00 74 0A 3C 20 74 06 3C 2F 75 E4 4E
[SU5] 669: 41 4A 75 D3 8D 55 FF 2B D5 88 56 00 1F FF A6 FC
[3T5] 670: 00 B0 0D E3 02 49 AC 3C D0 C3 BA 52 0A FF E2 BA
[DV5] 671: 45 0A FF E2 8B EC FF 76 04 FF 76 02 BA D2 0A FF
[E06] 672: D2 8B EC FF 76 08 FF 76 06 BA 45 0A FF D2 BA BE
[HT5] 673: 06 FF D2 8B EC 8F 46 06 8F 46 08 C3 8B EC 8B 46
[VE5] 674: 04 24 7F 80 FC 81 72 12 BB 00 81 53 33 DB 53 50
[NZ5] 675: FF 76 02 BA BE 06 FF D2 BB 04 50 FF 76 02 8B EC
[2S5] 676: FF 76 02 FF 76 00 FF 76 02 FF 76 00 BA 22 06 FF
[KU5] 677: D2 B8 B3 76 50 B8 BD 83 50 8B EC FF 76 06 FF 76
[YR5] 678: 04 BA 22 06 FF D2 B8 1E 79 50 B8 A6 F4 50 BA 86
[9W5] 679: 05 FF D2 8B EC FF 76 06 FF 76 04 BA 22 06 FF D2
[MW5] 680: B8 83 7B 50 B8 B0 FC 50 BA 86 05 FF D2 8B EC FF
[NR5] 681: 76 06 FF 76 04 BA 22 06 FF D2 B8 0C 7C 50 B8 67
[LW5] 682: 1F 50 BA 86 05 FF D2 8B EC FF 76 06 FF 76 04 BA
[3S5] 683: 22 06 FF D2 B8 DE 7C 50 B8 CB 53 50 BA 86 05 FF
[DU5] 684: D2 8B EC FF 76 06 FF 76 04 BA 22 06 FF D2 B8 14
[WS5] 685: 7D 50 B8 70 64 50 BA 86 05 FF D2 8B EC FF 76 06
[TR5] 686: FF 76 04 BA 22 06 FF D2 B8 B7 7D 50 B8 51 EA 50
[EV5] 687: BA 86 05 FF D2 8B EC FF 76 06 FF 76 04 BA 22 06
[2U5] 688: FF D2 B8 63 7D 50 B8 88 30 50 BA 86 05 FF D2 8B
[FS5] 689: EC FF 76 06 FF 76 04 BA 22 06 FF D2 B8 92 7E 50
[6W5] 690: B8 99 44 50 BA 86 05 FF D2 8B EC FF 76 06 FF 76
[YS5] 691: 04 BA 22 06 FF D2 B8 4C 7E 50 B8 91 CC 50 BA 86
[9W5] 692: 05 FF D2 8B EC FF 76 06 FF 76 04 BA 22 06 FF D2
[8Y5] 693: B8 AA 7F 50 B8 AA AA 50 BA 86 05 FF D2 8B EC FF
[2H5] 694: 76 06 FF 76 04 BA 22 06 FF D2 B8 01 50 33 C0
[YR5] 695: 50 BA 86 05 FF D2 8B EC 8F 46 04 8F 46 06 BA 22
[1L5] 696: 06 FF D2 8B EC 80 7E 09 81 72 19 80 7E 03 00 74
[2F5] 697: 04 80 76 02 80 B8 49 81 50 B8 DA 0F 50 BA 86 05
[FM5] 698: FF D2 8B EC F6 46 08 80 74 0A 80 7E 03 00 74 04
[YP3] 699: 80 76 02 80 8F 46 06 8F 46 08 C3
```

Listing 3. Die assemblierte Laufzeitbibliothek »basic.bin« als Hexdump

Toolboxen für Programmierer gesucht

Haben Sie eine Toolbox in C, Pascal, Basic, dBase, Clipper oder Assembler geschrieben? Beispielsweise schnelle, universelle Grafikroutinen, eine Textverarbeitungs- oder eine Datenbank-Toolbox? Dann sollten Sie sich schnellstmöglich mit uns in Verbindung setzen. Schicken Sie uns unverbindlich eine Diskette mit Ihrem Programm und einen Ausdruck der Dokumentation.

Falls Sie sich vorweg informieren wollen, zwischen 10.00 und 12.00 Uhr und zwischen 15.00 und 17.00 Uhr stehen wir Ihnen an allen Werktagen unter der Rufnummer (089) 4 61 00 10 zur Verfügung.

Schicken Sie Ihre Unterlagen an folgende Adresse:

Redaktionsbüro Everts & Hagedorn

»DOS Software«

Hans-Stießberger-Str. 3

8013 Haar/München

Impressum

Herausgeber

Christian Widuch

Leitung des Redaktionsbüros

Volker Everts (ev), Andreas Hagedorn (hg)

Chefredakteur

Martin Aschoff (ma)

Redaktion

Gabi Börger (gb), Werner Breuer (wb), Ulli Eike (ue), Johann Glas (gh),
Eva-Maria Hierlmeier (hi), Alois Kneisle (al), Manfred Kohlen (mk),
Thomas Röder (tr), Andrea Scheduling (as), Stefan Willmeroth (sw),
Johann Zieger (zi)

Schlußredaktion

Karl-Heinz Peter (kp), Markus Stein (ms), Andrea Voß (av)

Redaktionsassistenten

Dorothea Lehner

Koordination

Stefan Ritter

Produktionsleitung

Gerd Köberich, Helmut Skoupy

Layout

Yvonne Hendricks, Patricia Reifenhausen, Michael Grebenstein

Satz

Claudia Küllmer, Silvia Führer, Martina Siebert,
Gabriela Joseph, Marcus Geppert, Heidemarie Kohlhaas

Lektorat

Susanne Mias, Dagmar Wilhelm

Reprografie und Gestaltung

Manuela Eska, Margarete Schenk, Dieter Schnobl,
Silvia Erbrich, Andrea Gundlach

Werbegestaltung

Mohamed Hawa, Petra Biehl

Fotografie

Christian Heckmann, Klaus Jatho

Bildarchiv

Heike Meister

Anzeigenverkaufsleitung

Wolfgang Schnell

Anzeigenverkauf für PLZ 1 - 5

Gerlinde Rachow (0 56 51) 80 09 - 53

Wolfgang Brill (0 56 51) 80 09 - 51

Anzeigenverkauf für PLZ 6 - 8

DMV-Verlagsbüro München

Zaunkönigweg 2c, 8000 München 82,

Telefon: (0 89) 4 39 10 87, Telefax: (0 89) 4 39 10 80

Leitung: Britta Fiebig

Anzeigenverkauf: Monika Schöbel, Jens Dhein

Anzeigenverwaltung und Disposition

Andrea Giese, Karina Ehrlich, Susanne Eska

Anschrift Redaktion:

Redaktionsbüro Everts & Hagedorn

Redaktion DOS International,

Hans-Stießberger-Str. 3, 8013 Haar/München

Telefon: (0 89) 46 10 01-0

Telefax: (0 89) 46 10 01-90

Anschrift Verlag:

DMV Daten und Medien Verlag

Widuch GmbH & Co. KG

Fuldaer Straße 6, 3440 Eschwege

Telefon: (0 56 51) 80 09-0

Telefax: (0 56 51) 80 09-33

Vertrieb

Verlagsunion Erich Pabel-Arthur Moewig KG (VPM)

Friedrich-Bergius-Straße 20

6200 Wiesbaden

Druck

Druckerei Jungfer, 3420 Herzberg

Bezugspreise

DOS EXTRA erscheint viermal jährlich.

Einzelpreis DM 18,-, 145 Ös, 18,- sfr.

Bankverbindungen:

Postscheck Frankfurt/M: Kto.-Nr.: 23043-608

Raiffeisenbank Eschwege:

BLZ: 522 603 85, Kto.-Nr.: 245 7008

Manuskripte, Programme:

bitte an das Redaktionsbüro Everts & Hagedorn,
Redaktion DOS International, Hans-Stießberger-
Straße 3, 8013 Haar/München senden.

Für unverlangt eingesandte Manuskripte und Datenträger sowie Fotos über-
nehmen der Verlag und die Redaktion keine Haftung. Die Zustimmung zum
Abdruck wird vorausgesetzt.

Das Urheberrecht für veröffentlichte Manuskripte liegt ausschließlich beim
Verlag. Nachdruck sowie Vervielfältigung oder sonstige Verwertung von Texten
nur mit schriftlicher Genehmigung des Verlages. Für veröffentlichte Program-
me und Basteleien übernehmen wir weder Gewähr noch Haftung. Schaltun-
gen und verwendete Bezeichnungen müssen nicht frei von gewerblichen
Schutzrechten sein.

Namentlich gekennzeichnete Fremdbeiträge geben nicht in jedem Fall die
Meinung der Redaktion wieder.

Die Inserenten

Bitter, Ing.-Büro.....	9
Computer Solution.....	172
CWTG.....	59
DMV.....	2,15,45,65,107,117,121,171
Interest-Verlag	51
Kayser-Datentechnik	35
Kirschbaum.....	41
MCS.....	49
Zoschke.....	109

Wer – wo – wann?

Bei Zuschriften bitten wir, folgendes zu
beachten: Haben Sie Fragen,
Anregungen oder Programme, die Sie
der DOS-Redaktion zukommen
lassen möchten, adressieren Sie wie folgt:

Redaktion

DOS International

Hans-Stießberger-Str. 3

8013 Haar/München

Tel. (089) 46 10 01-0

Haben Sie Fragen, die zum Beispiel
Abonnements, Kleinanzeigen oder
Databox-Bestellungen betreffen, wenden
Sie sich bitte an den Verlag:

DMV-Verlag

DOS International

Postfach 250

3440 Eschwege

Telefon (0 56 51) 80 09-0

Telefax (0 56 51) 80 09-33

Für jeden Computerfreund etwas dabei

DMV-Zeitschriften



Joystick

Alles was Computern Spaß macht

3

Hot-Shot
Public Domain
Kapitel: Platte
Warten: die
Pilot List
Captain Power

Das Magazin für alle
geringen Systemen

DOS

INTERNATIONAL

3

Mehr power
Sensationell:
eigener dreschender
Sonne selbst
programmieren

Das MAGAZIN FÜR ANFÄNGER
ANWENDER
PROGRAMMIERER
UND
PROFIS

DEMO
IMHEIT
DISKETTE

Window

2

Das Magazin zur ausstrahlenden Software

Benutzeroberflächen
Erforderliche
FRAGEN IN
GEGEN ENKLE
Kleiner Computer
WIDENSTAR 5.0

Mehr
Durchschnitt
- 100
- 100
- 100

BROSSE LESENUMFRAGE
Gewinnen Sie mit
Window!

Toolbox

3

Software verstehen und anwenden

Top-Listings:
- 100
- 100
- 100

Alles über mouse
- 100
- 100
- 100

Referenzen zum Sammeln
- 100
- 100
- 100

Super
grafik
programm

Amstrad

INTERNATIONAL

3

Joyce/PCW:
- 100
- 100
- 100

Alles im Griff:
- 100
- 100
- 100

Super
PC:
- 100
- 100
- 100

JOYSTICK
Siele ohne JOYSTICK sind
wie Pacman ohne Geister!
Das Unterhaltungsmagazin.

PC pur
Einstiege und Grundlagen
müssen verständlich sein
PC-Neulinge wissen das zu
schätzen.

DOS International
Lesen Sie "DOS" - liegen
Sie richtig Für Anwender
und Programmierer aller
PCs.

WINDOW
Anspruchsvolle Software
setzt professionelles Arbeit
ten voraus. Das Magazin
das neue Maßstäbe setzt.

PC Amstrad International
Die Nummer Eins, wenn es
um CPCs, PCW Joyce und
PC 1512/1640 geht.

TOOLBOX
Die Fachzeitschrift für Pro-
grammierer. Alternative für
alle modernen Sprachen der
modernen Computersy-
steme.

Telefon: ()
Datum: . 1989